SuperH RISC engine

SH7040 Series
On-Chip Supporting Modules
Application Note

HITACHI

Name
Hitachi Micro Systems, Incorporated
1/24/97

Notice

When using this document, keep the following in mind:

1. This document may, wholly or partially, be subject to change without notice.
2. All rights are reserved: No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without Hitachi's permission.
3. Hitachi will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's unit according to this document.
4. Circuitry and other examples described herein are meant merely to indicate the characteristics and performance of Hitachi's semiconductor products. Hitachi assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.
5. No license is granted by implication or otherwise under any patents or other rights of any third party or Hitachi, Ltd.
6. MEDICAL APPLICATIONS: Hitachi's products are not authorized for use in MEDICAL APPLICATIONS without the written consent of the appropriate officer of Hitachi's sales company. Such use includes, but is not limited to, use in life support systems. Buyers of Hitachi's products are requested to notify the relevant Hitachi sales offices when planning to use the products in MEDICAL APPLICATIONS.

Preface

The SH7040, SH7041, SH7042, and SH7043 are high-performance microcomputers with a 32-bit SH-2 CPU core that uses a RISC (reduced instruction set computer) type instruction set, and comprehensive on-chip peripheral functions.

On-chip peripherals include ROM, RAM, a 16-bit multifunction timer pulse unit (MTU), serial communication interface (SCI), port output enable (POE), data transfer controller (DTC), and DMA controller (DMAC), enabling these microcomputers to be used for a wide range of applications covering small to large-scale systems.

This Application Note includes sample tasks that use the SH7040 Series' on-chip peripheral functions, which we hope users will find useful in carrying out software design.

The operation of the task programs in this Application Note has been checked, but operation should be confirmed again before any of these programs are actually used.

Contents

Section 1 Using the SH7040 Series Application Note

This Application Note describes the peripheral functions of the SH7040 Series by means of simple sample tasks.

1.1 Application Note Organization

The on-chip I/O volume uses the layout shown in figure 1 to describe the use of the peripheral functions.

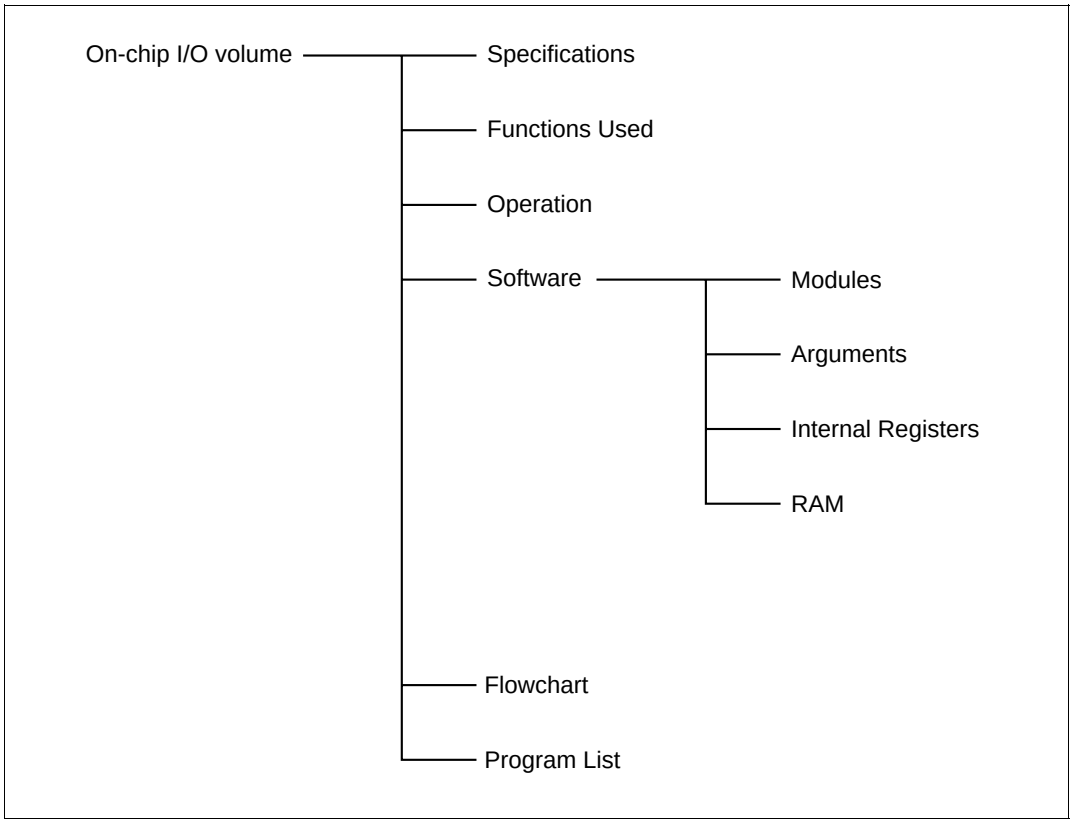


Figure 1 Application Note Organization

Specifications

Describes the system specifications for each task.

Functions Used

Describes the features of the peripheral function(s) used in the sample task, and peripheral function assignment.

Operation

Describes the operation of each task, using timing charts.

Software

1. **Modules**
   Describes the software modules used in the operation of the sample task.
2. **Arguments**
   Describes the input arguments needed to execute the module, and the output arguments after execution.
3. **Internal Registers**
   Describes the peripheral function internal registers (timer control registers, serial mode registers, etc.) set by the modules.
4. **RAM**
   Describes the labels and functions of the RAM used by the modules.

Flowchart

Describes the software that executes the sample task, using a general flowchart.

Program List

Shows a program list of the software that executes the sample task.

2.1 Pulse High and Low Width Measurement MTU (Input Capture)

Specifications

1. Pulse high- and low-level widths are measured as shown in figure 2-1-1, and stored in RAM.
2. At 28.7 MHz operation, pulse high- and low-level widths of 35 ns to 2.28 ms can be measured, in 35 ns units.

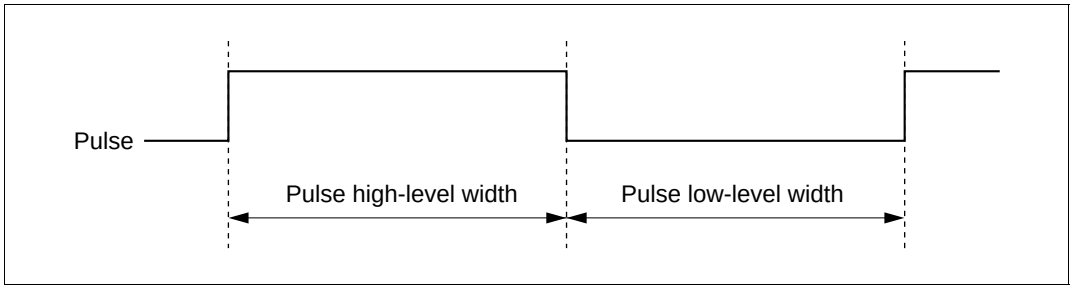


Figure 2-1-1 Pulse Width Measurement Timing

Functions Used

1. In this sample task, pulse high- and low-level widths are measured using ch0.

   a. Figure 2-1-2 shows the ch0 block diagram. This task uses the following functions.

   - A function that performs detection of the rising edge and falling edge of a pulse, and sets the timer value at that point in an internal register (input capture)
   - A function that clears the timer counter when input capture occurs (counter clear)
   - A function that initiates interrupt handling on detection of the rising edge and falling edge of a pulse

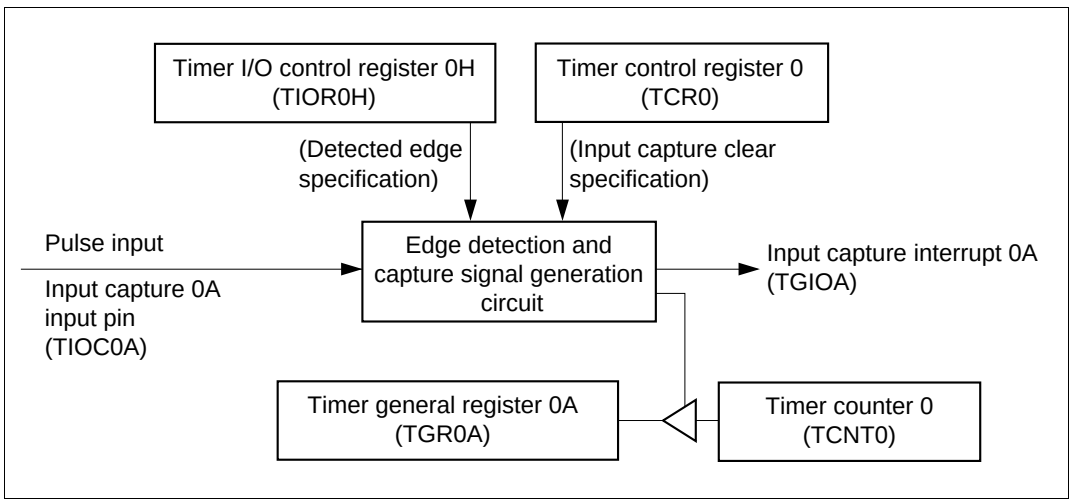


Figure 2-1-2 MTU/ch0 Block Diagram

2. Table 2-1-1 shows the function assignments for this sample task. MTU functions are assigned as shown in this table to measure pulse high- and low-level widths.

Table 2-2-1 Function Assignment

Pin/Register Name	Function Assignment
TCR0	Selects counter clear source
TIOR0H	Selects input capture signal input edge
TIOC0A	Inputs pulse to be measured
TGR0A	Detects counter value at rise and fall of pulse
TGI0A	Initiates pulse high- and low-level width measurement on rise and fall of pulse

Operation

Figure 2-1-3 shows the principles of the operation. Pulse high- and low-level widths are measured by means of SH7040 Series hardware and software processing as shown in the figure.

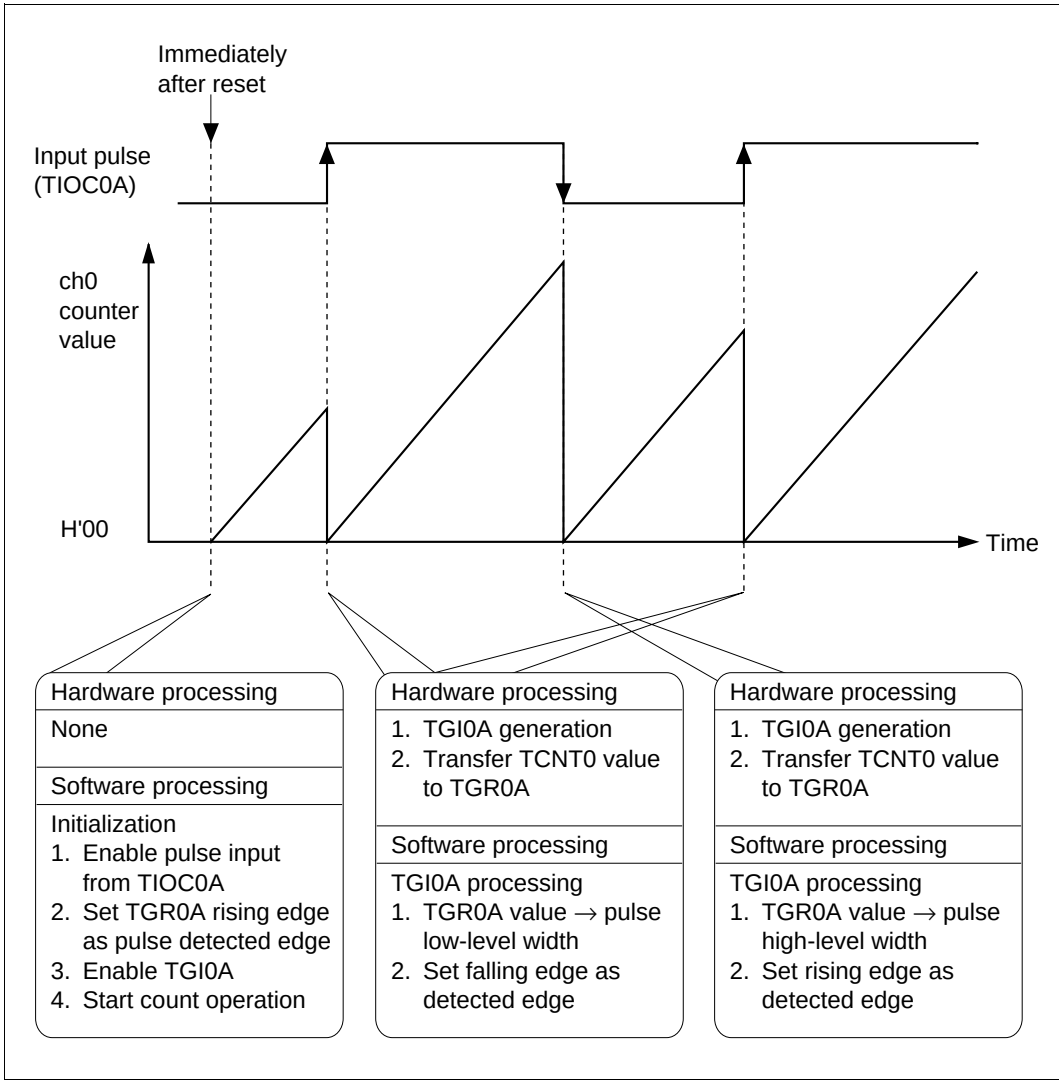


Figure 2-1-3 Principles of Pulse Width Measurement Operation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	pwhlmn	MTU initialization
Pulse high- and low-level width measurement	pwhl1	Initiated by TGI0A; measures pulse high- and low-level widths according to TGR0A value and sets result in RAM

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
pwh_hdata	Setting of timer value corresponding to pulse high-level width Pulse high-level width is calculated from following formula: Pulse high-level width (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	1 word	Pulse high- and low-level width measurement	Output
pwh_ldata	Setting of timer value corresponding to pulse low-level width Pulse low-level width is calculated from following formula: Pulse low-level width (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	1 word		

3. Internal Registers Used

Register Name	Function Assignment	Module
PFCE.PECR2	Enables pulse input from TIOC0A	Main routine
T0.TCR0	Selects TCNT counter clock, specifies input capture A as counter clear source	Main routine Pulse high- and low-level width measurement
T0.TIOR0H	Set so that transfer is performed from TCNT to TGR0A on detection of pulse rise or fall	Main routine
T0.TIER0	Enables interrupts by TGI0A	
T0.TGRA0	TCNT value at rise or fall of pulse is set in this register and used to calculate pulse cycle	Pulse high- and low-level width measurement
T0.TSR0	Input capture A generation status	
INTC.IPRD	Sets TGIOA interrupt priority level to 15	Main routine
TMRSH.TSTR	Specifies timer counter operation/disabling	

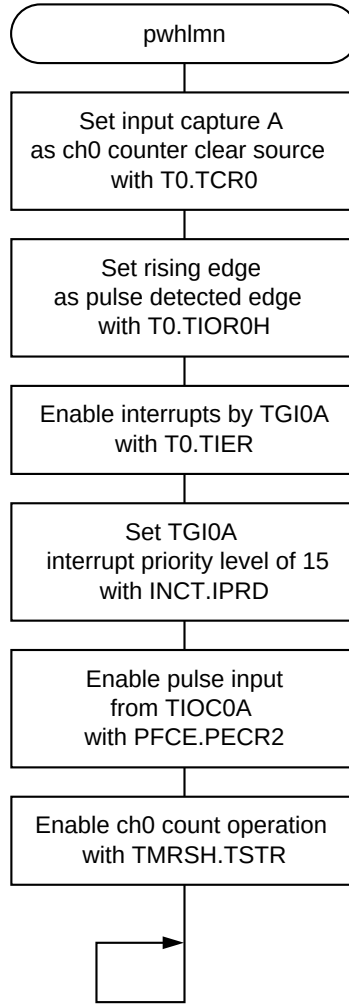
4. RAM Used

This sample task does not use any RAM apart from the arguments.

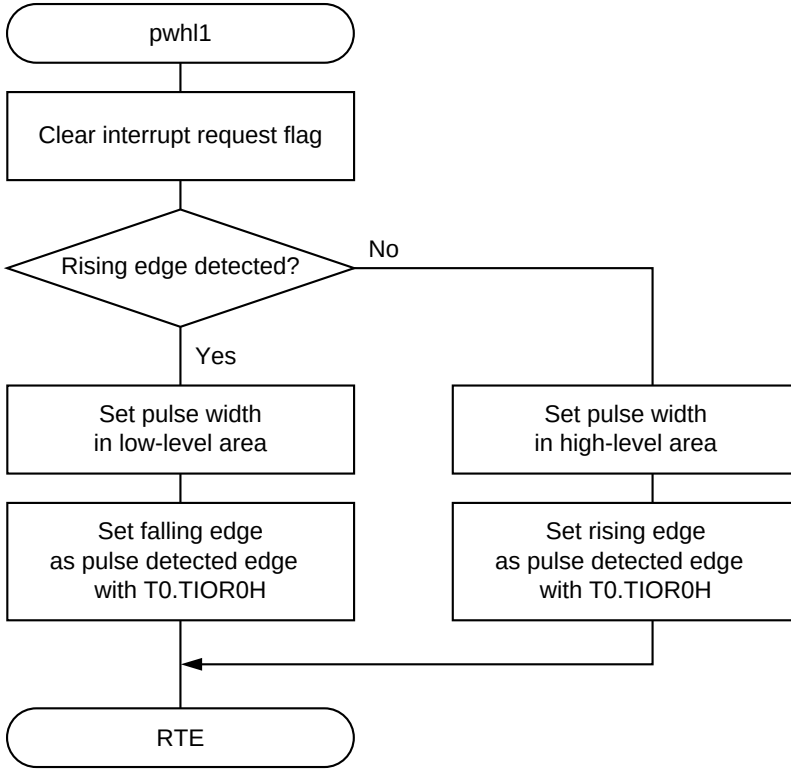
Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



2. Pulse high- and low-level width measurement



Program List

```
/*------------------------------------------------------------------------*/
/*                                INCLUDE FILE                            */
/*------------------------------------------------------------------------*/
#include <machine.h>
#include "SH7040.H"
/*------------------------------------------------------------------------*/
/*                                PROTOTYPE                               */
/*------------------------------------------------------------------------*/
void pwhlmn(void);
#pragma interrupt(pwhl1)
/*------------------------------------------------------------------------*/
/*                              RAM ALLOCATION                            */
/*------------------------------------------------------------------------*/
#define pwh_hdata      (*(unsigned short *)0xffffe800)
#define pwh_ldata      (*(unsigned short *)0xffffe802)
/*------------------------------------------------------------------------*/
/*                                MAIN PROGRAM                            */
/*------------------------------------------------------------------------*/
void pwhlmn(void)
{
    T0.TCR0 = 0x20;                  /* timer clear input capture TGRA0 */
    T0.TIOR0H = 0x08;                /* input capture rising edge TIOC0A */
    T0.TIER0 = 0x01;                 /* initialize TGIOA0 */
    INTC.IPRD = 0xf000;              /* set initialize level=15 */
    set_imask(0x0);                  /* set imask level=0 */
    PFCE.PECR2 = 0X0001;             /* TIOCnx select */
    TMRSH.TSTR = 0x01;               /* start timer0 */
    while(1);                        /* loop */
}

void pwhl1()
{
    T0.TSR0 =0xfe;                   /* clear flag */
    if(( T0.TIOR0H & 0x0f ) == 0x08)   /* rising edge? */
    {
    pwh_hdata = T0.TGR0A;            /* set pwh */
    T0.TIOR0H |= 0x01                /* input capture falling edge TIOC0A */
    }
    else
    {
    pwh_ldata = T0.TGR0A;            /* set pwl */
    T0.TIOR0H &=0xfe;                /* input capture rising edge TIOC0A */
```

Specifications

1. Using MTU ch0, a 50% duty pulse of the specified cycle is output as shown in figure 2-2-1.

2. At 28.7 MHz operation, any output pulse cycle from 69.6 ns to 2.28 ms can be set.

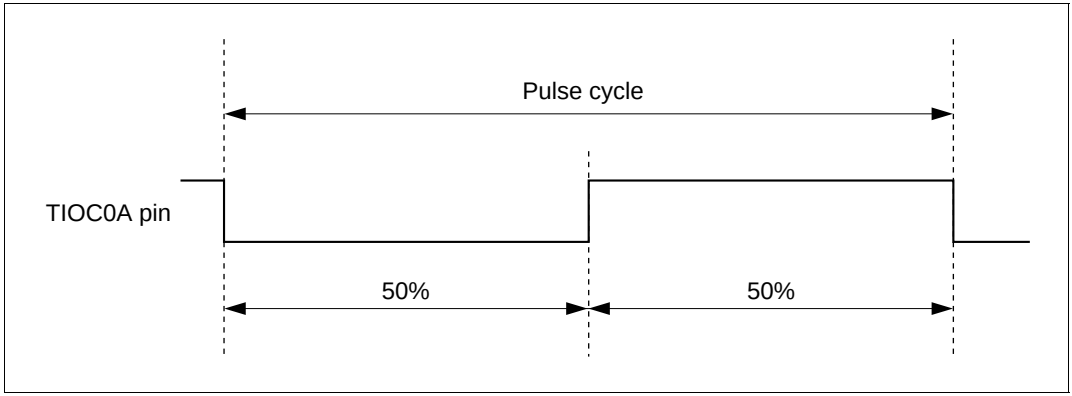


Figure 2-2-1 Example of Pulse Output

Functions Used

1. In this sample task, a pulse with a 50% duty cycle is output using MTU ch0.

   a. Figure 2-2-2 shows the MTU/ch0 block diagram for by this sample task. The following functions are used by ch0:

      - A function for automatically outputting pulses by hardware without software intervention (output compare)
      - A function that clears the timer counter in the event of a compare-match (counter clear)
      - A function that inverts the output each time a compare-match occurs (toggle output)

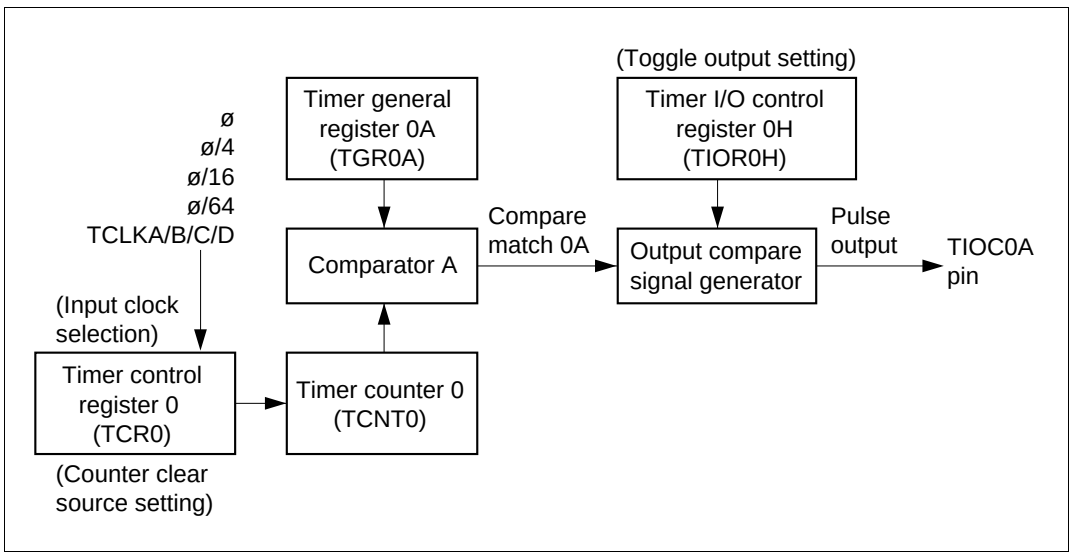


Figure 2-2-2 MTU/ch0 Block Diagram

2. Table 2-2-1 shows the function assignments for this sample task. MTU functions are assigned as shown in this table to perform pulse output.

Table 2-2-1 Function Assignment

Pin/Register Name	Function Assignment
TIOC0A	Pulse output pin
TCR0	Selects counter clear source and input clock
TIOR0H	Pulse output level setting
TGR0A	1/2 pulse cycle setting

Operation

Figure 2-2-3 shows the principles of the operation. Pulses are output by means of SH7040 hardware and software processing as shown in the figure.

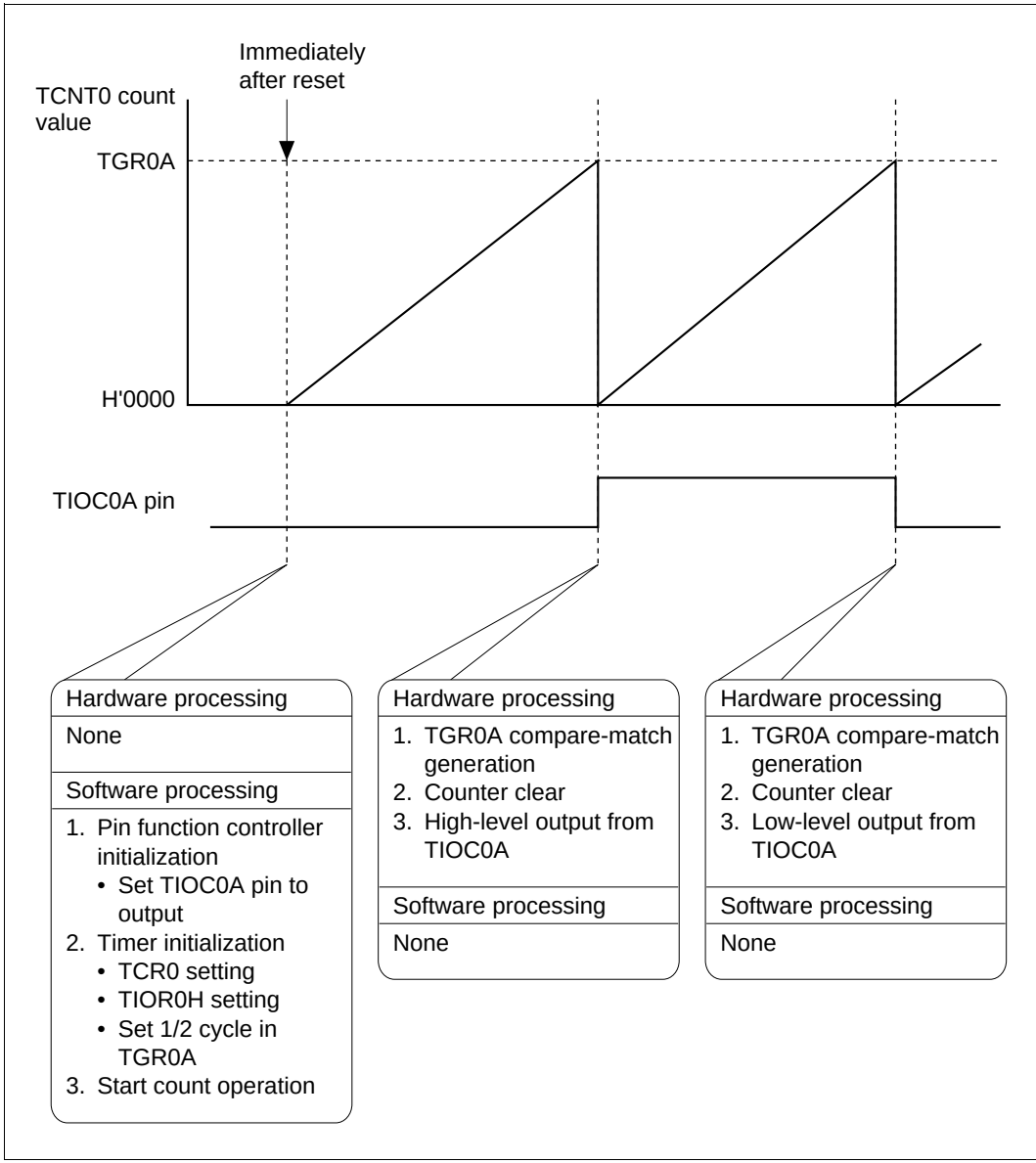


Figure 2-2-3 Principles of Pulse Output Operation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	puls_out	PFC and pulse output setting

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
pul_cyc	Setting of timer value corresponding to 1/2 pulse cycle Pulse cycle is calculated from following formula: Pulse cycle (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	1 word	Main routine	Input

3. Internal Registers Used

Register Name	Function Assignment	Module
PFCE.PECR2	Sets multiplexed pin to TIOC0A output	Main routine
TMRSH.TSTR	Sets ch0 timer counter operation/disabling	
T0.TCR0	Specifies TGR0A compare-match as counter clear source, and ø as input clock	
T0.TIOR0H	Initial TIOC0A output = 0; output toggled on compare-match	
T0.TGR0A	1/2 output pulse cycle setting	
T0.TMDR0	Sets ch0 to normal mode	

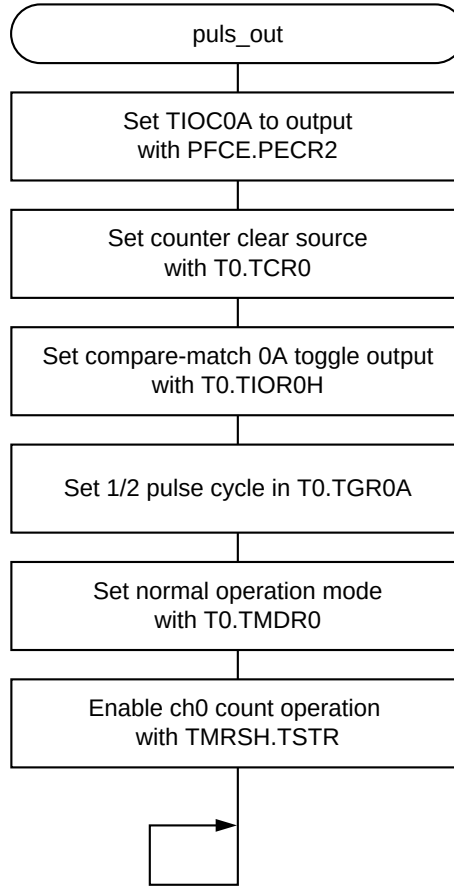
4. RAM Used

This application example does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



Program List

```
/*------------------------------------------------------------*/
/*                        INCLUDE FILE                        */
/*------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*------------------------------------------------------------*/
/*                         PROTOTYPE                          */
/*------------------------------------------------------------*/
void puls_out(void);
/*------------------------------------------------------------*/
/*                       RAM ALLOCATION                       */
/*------------------------------------------------------------*/
#define pul_cyc        (*(unsigned short *)0xffffe800)
/*------------------------------------------------------------*/
/*                        MAIN PROGRAM                        */
/*------------------------------------------------------------*/
void puls_out(void)
     {
     PFCE.PEIOR = 0x0001;      /* TIOC0A output */
     PFCE.PECR2 = 0x0001;      /* TIOC0A output compare/match output */

     T0/TCR0 = 0x20;           /* compare/match clear of TGR0A */
     T0.TIOR0H = 0x03;         /* start' 0' compare/match toggle output */
     T0.TGR0A = pul_cyc;       /* 1/2 sycle */
     T0.TCNT0 = 0x0000;        /* timer start value set */
     T0.TMDR0 = 0xc0;          /* mode set */
     TMRSH.TSTR = 0x01;        /* timer count start */
     while(1);
     }
```

Specifications

1. Using MTU PWM mode 1, 4-phase PWM output is performed based on the specified duty values and cycles.

2. PWM mode 1 allows any cycle to be set for each channel. Two outputs are possible for each of ch0, ch3, and ch4, and one output for ch1 and ch2. Thus waveforms with different high-level widths can be generated in the same cycle on ch0, ch3, and ch4.

3. A duty of 0% to 100% can be set, with 1/65535 resolution.

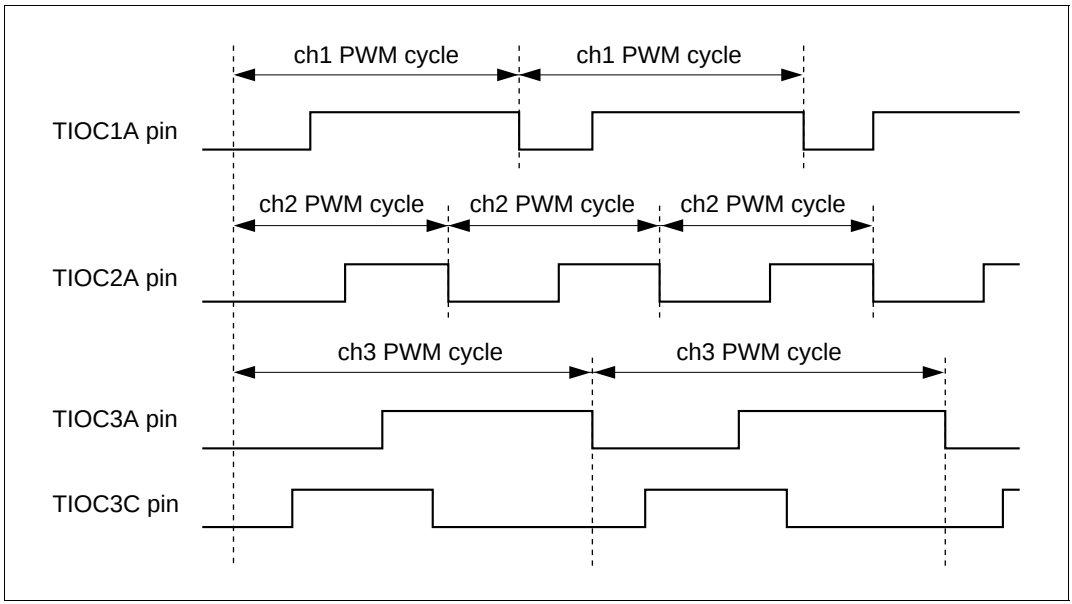


Figure 2-3-1 Example of PWM Output

Functions Used

1. In this sample task, 4-phase PWM output is performed using MTU ch1 to ch3.

   In PWM mode 1, PWM output is generated using the TGRA and TGRB registers as a pair, and the TGRC and TGRD registers as a pair. Up to 8-phase PWM output is possible by using ch0 to ch4.

   a. Figure 2-3-2 shows the MTU block diagram for this sample task.

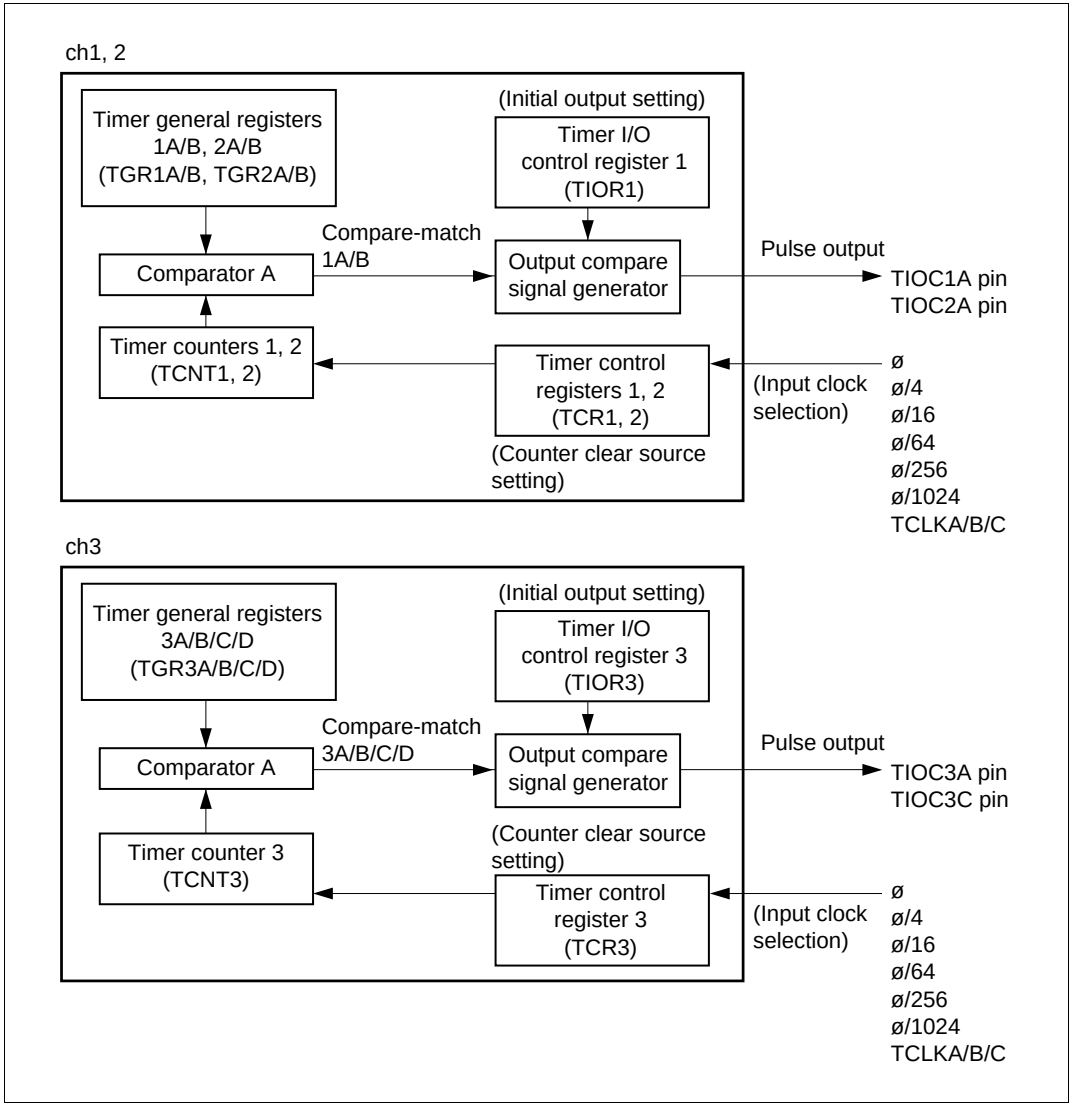


Figure 2-3-2 MTU/ch1, ch2, ch3 Block Diagram

2. Table 2-3-1 shows the function assignments for this sample task. MTU functions are assigned as shown in this table to perform PWM pulse output.

Table 2-3-1 Function Assignment

Pin/Register Name	Function Assignment
TIOC1A TIOC2A TIOC3A TIOC3C	PWM pulse output pins
TCR1 TCR2 TCR3	Select ch1 to ch3 timer counter clear sources and input clocks
TMDR1 TMDR2 TMDR3	Specify ch1 to ch3 operating in PWM mode 1
TGR1A TGR2A TGR3A	PWM cycle settings
TGR1B TGR2B TGR3B TGR3C TGR3D	Duty value settings

Operation

Figure 2-3-3 shows the principles of the operation. 4-phase PWM output is performed from the ch1 to ch3 PWM output pins (TIOC1A, TIOC2A, TIOC3A/C) by means of SH7040 hardware and software processing as shown in the figure.

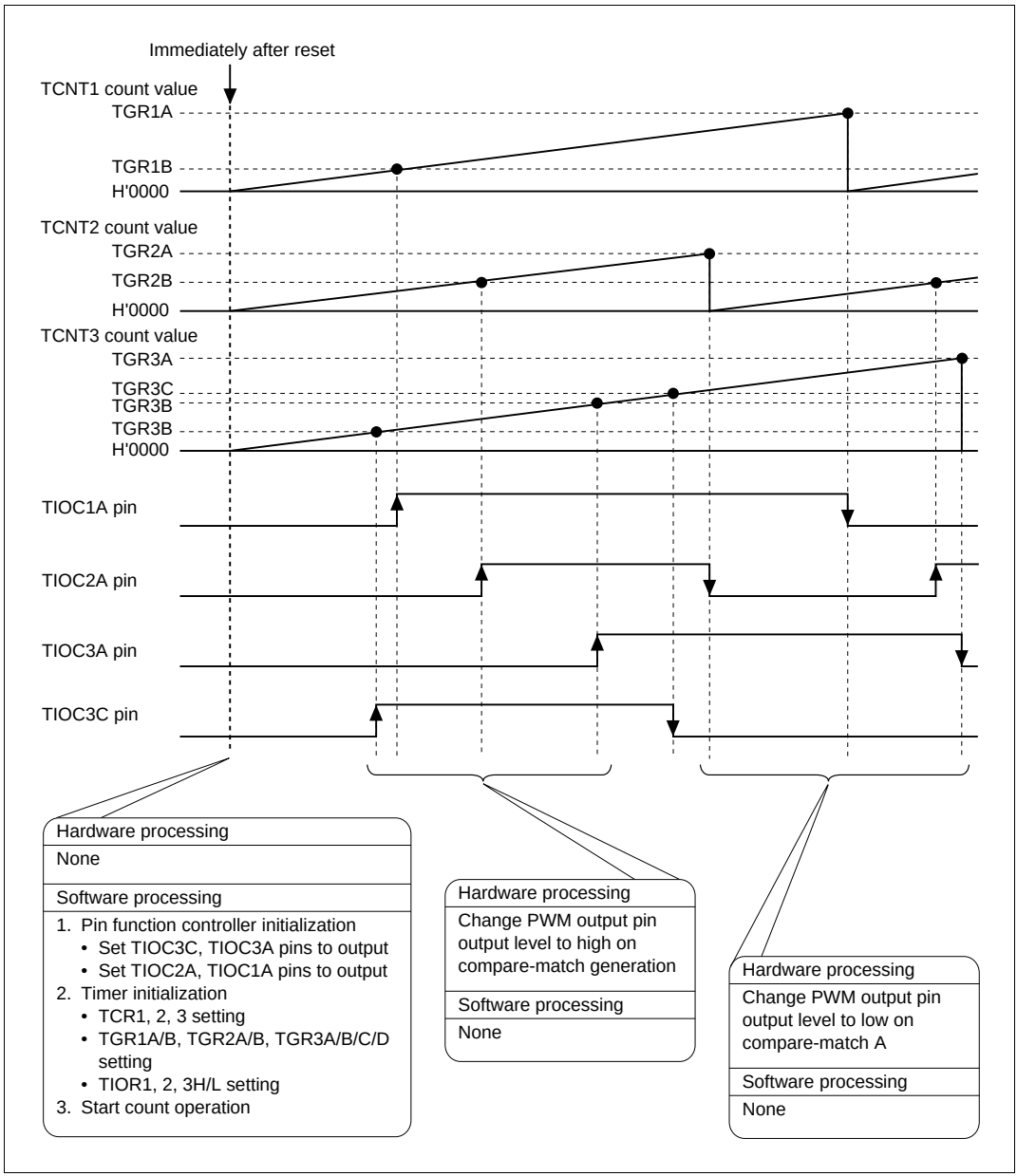


Figure 2-3-3 Principles of PWM Waveform Operation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	pwm_1	PFC and PWM output setting

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
pul_cyc1 pul_cyc2 pul_cyc3	Setting of timer value corresponding to pulse cycle Pulse cycle is calculated from following formula: Pulse cycle (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	1 word	Main routine	Input
pul_duty1b pul_duty2b pul_duty3b pul_duty3c pul_duty3d	Setting of timing for change of waveform output from TIOC pin			

3. Internal Registers Used

Register Name	Function Assignment	Module
TMRSH.TSTR	Performs start of channel 1, 2, 3, timer count	Main routine
T1.TCR1 T2.TCR2 T3.TCR3	Clear timer counter clear source on TGR1A, TGR2A, TRG3A compare-match Select ø as input clock	
T1.TGR1A	Channel 1 PWM cycle setting	
T1.TGR1B	Setting of timer counter value that causes high-level output from TIOC1A	
T2.TGR2A	Channel 2 PWM cycle setting	
T2.TGR2B	Setting of timer counter value that causes high-level output from TIOC2A	
T3.TGR3A	Channel 3 PWM cycle setting	
T3.TGR3B	Setting of timer counter value that causes high-level output from TIOC3A	
T3.TGR3C	Setting of timer counter value for low-level output from TIOC3C	
T3.TGR3D	Setting of timer counter value that causes high-level output from TIOC3C	
T1.TIOR1	Sets 0 initial output and 0 output on output compare for TGR1A, and 0 initial output and 1 output on output compare for TGR1B	
T2.TIOR2	Sets 0 initial output and 0 output on output compare for TGR2A, and 0 initial output and 1 output on output compare for TGR2B	
T3.TIOR3H	Sets 0 initial output and 0 output on output compare for TGR3A, and 0 initial output and 1 output on output compare for TGR3B	
T3.TIOR3L	Sets 0 initial output and 0 output on output compare for TGR3C, and 0 initial output and 1 output on output compare for TGR3B	
T1.TMDR1 T2.TMDR2 T3.TMDR3	Set operating mode to PWM mode 1	

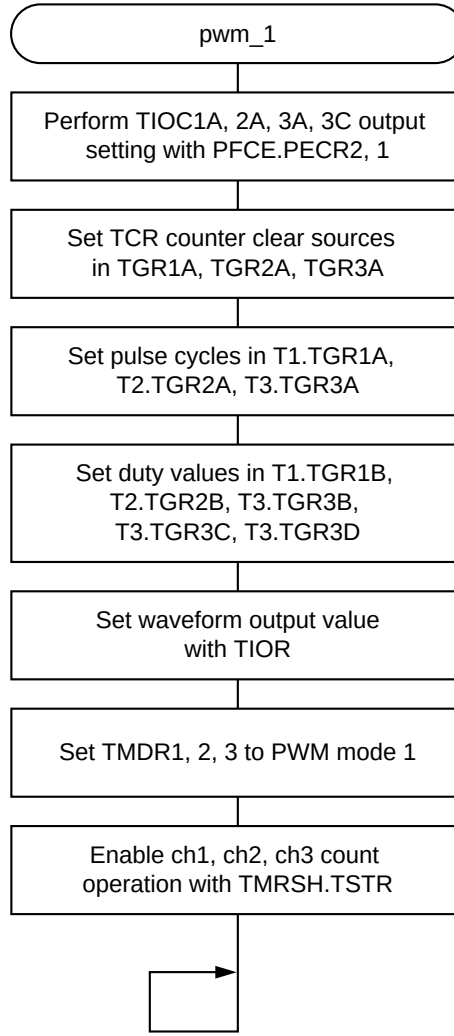
4. RAM Used

This application example does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



Program List

```
/*---------------------------------------------------------------------*/
/*                           INCLUDE FILE                              */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*---------------------------------------------------------------------*/
/*                            PROTOTYPE                                */
/*---------------------------------------------------------------------*/
void pwm_1(void);
/*---------------------------------------------------------------------*/
/*                         RAM ALLOCATION                              */
/*---------------------------------------------------------------------*/
#define pul_cyc1      (*(unsigned short *)0xffffe800)
#define pul_duty1b    (*(unsigned short *)0xffffe802)
#define pul_cyc2      (*(unsigned short *)0xffffe804)
#define pul_duty2b    (*(unsigned short *)0xffffe806)
#define pul_cyc3      (*(unsigned short *)0xffffe808)
#define pul_duty3b    (*(unsigned short *)0xffffe80a)
#define pul_duty3c    (*(unsigned short *)0xffffe80c)
#define pul_duty3d    (*(unsigned short *)0xffffe80e)
/*---------------------------------------------------------------------*/
/*                           MAIN PROGRAM                              */
/*---------------------------------------------------------------------*/
void pwm_1(void)
     {
     PFCE.PEIOR = 0x0550;     /* TIOC1A,TIOC2A,TIOC3A/C output */
     PFCE.PECR1 = 0x0011;     /* TIOC3A/C output */
     PFCE.PECR2 = 0x1100;     /* TIOC1A, TIOC2A output */

     T1.TCR1 = 0x20;          /* TGR1A compare/match clear */
     T1.TIOR1 = 0x02;         /* start '0' compare/match '1' output */
     T1.TCNT1 = 0x0000;       /* timer counter '0' set */
     T1.TGR1A = pul_cyc1;     /* timer general register set */
     T1.TGR1B = pul_duty1b;
     T1.TMDR1 = 0xc2;         /* pwm mode 1 */

     T2.TCR2 = 0x20;          /* TGR2A compare/match clear */
     T2.TIOR2 = 0x02;         /* start '0' compare/match '1' output */
     T2.TCNT2 = 0x0000;       /* timer counter '0' set */
     T2 TGR2A = pul_cyc2;     /* timer general register set */
     T2.TGR2B = pul_duty2b;
     T2.TMDR2 = 0xc2;         /* pwm mode 1 */

     T3.TCR3 = 0x20;          /* TGR3A compare/match clear */
     T3.TIOR3H = 0x02;        /* start '0' compare/match '1' output */
     T3.TIOR3L = 0x21;        /* start '0' compare/match '1' output */
     T31.TCNT3 = 0x0000;      /* timer counter '0' set */
     T31.TGR3A = pul_cyc3;    /* timer general register set */
     T31/TGR3B = pul_duty3b;
     T31.TGR3C = pul_duty3c;
     T31.TGR3D = pul_duty3d;
     T3.TMDR3 = 0xc2;         /* pwm mode 1 */

     TMRSH.TSTR = 0x46;       /* timer start */
     while(1);
     }
```

Specifications

1. The pulse high-level width is varied to generate variable-duty 7-phase PWM output as shown in figure 2-4-1.

2. At 28.7 MHz operation, any output PWM cycle from 69.9 ns to 2.28 ms can be set.

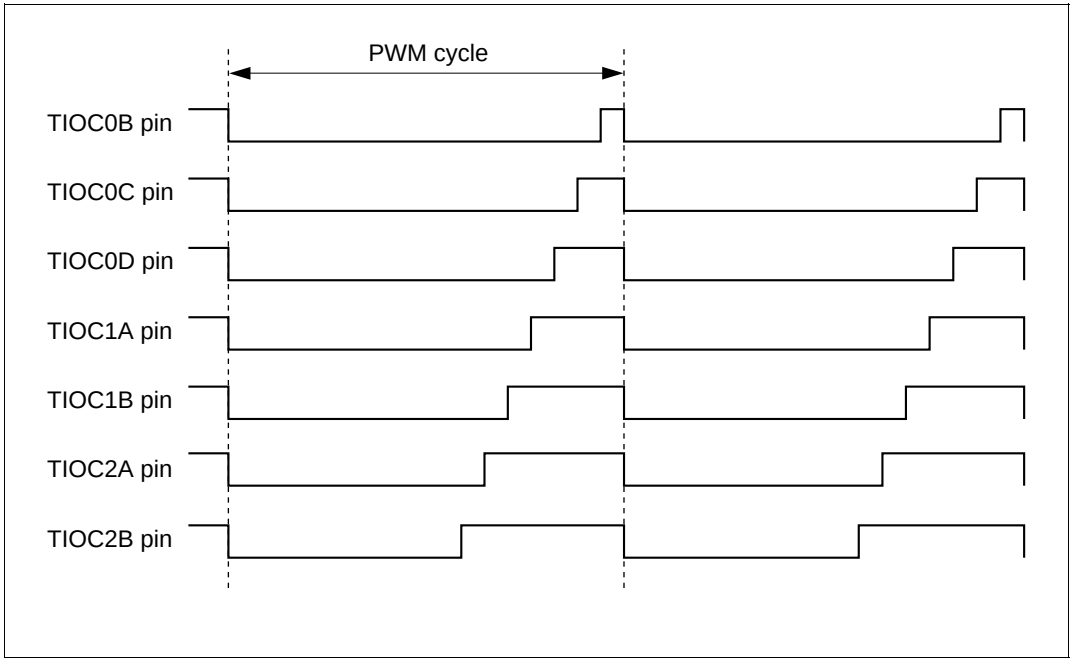


Figure 2-4-1 Example of PWM Output

Functions Used

1. In this sample task, 7-phase PWM output is performed by synchronous operation of MTU ch0 to ch2.

   a. Figure 2-4-2 shows the MTU block diagram for this sample task.

   This sample task uses the following MTU functions:

   - A function for automatically outputting pulses by hardware without software intervention (output compare-match)
   - A function that clears the counter in the event of a compare-match (counter clear)
   - A function that inverts the output each time a compare-match occurs (toggle output)

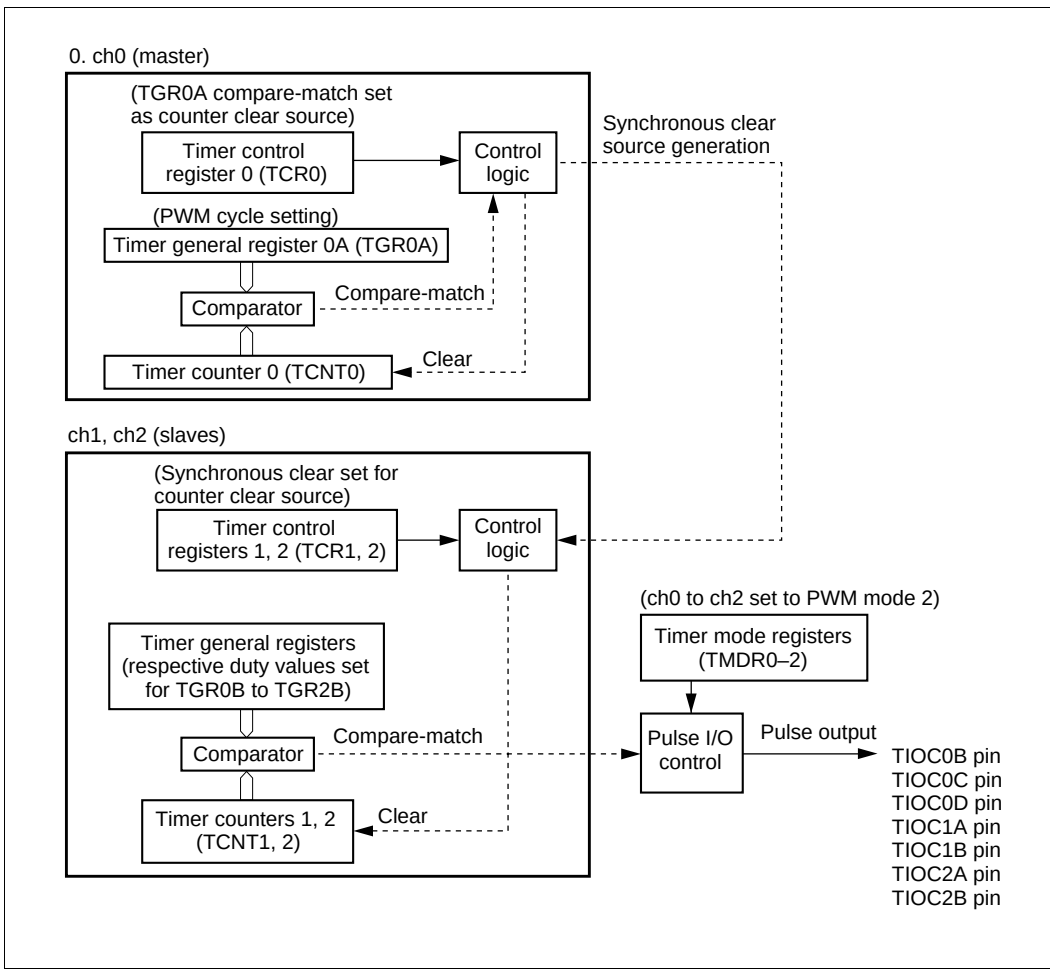


Figure 2-4-2 Synchronous Clear Block Diagram

2. Table 2-4-1 shows the function assignments for this task. MTU functions are assigned as shown in this table to perform PWM pulse output.

Table 2-4-1 MTU Function Assignment

Pin/Register Name	Function Assignment
TIOC0B TIOC0C TIOC0D TIOC1A TIOC1B TIOC2A TIOC2B	PWM pulse output pins
TSYR	Synchronous operation of ch0 to ch2
TCR0 to TCR2	Select ch0 to ch2 timer counter clear sources and input clocks
TGR0A	PWM cycle setting
TGR0B TBR0C TGR0D TGR1A TGR1B TGR2A TGR2B	Duty value settings
TMDR0 to TMDR2	Specify ch0 to ch2 operating in PWM mode 2

Operation

Figure 2-4-3 shows the principles of the operation. 7-phase PWM output is performed from the ch0 to ch2 PWM output pins (TIOC0B/C/D, TIOC1A/B, TIOC2A/B) by means of SH7040 hardware and software processing as shown in the figure.

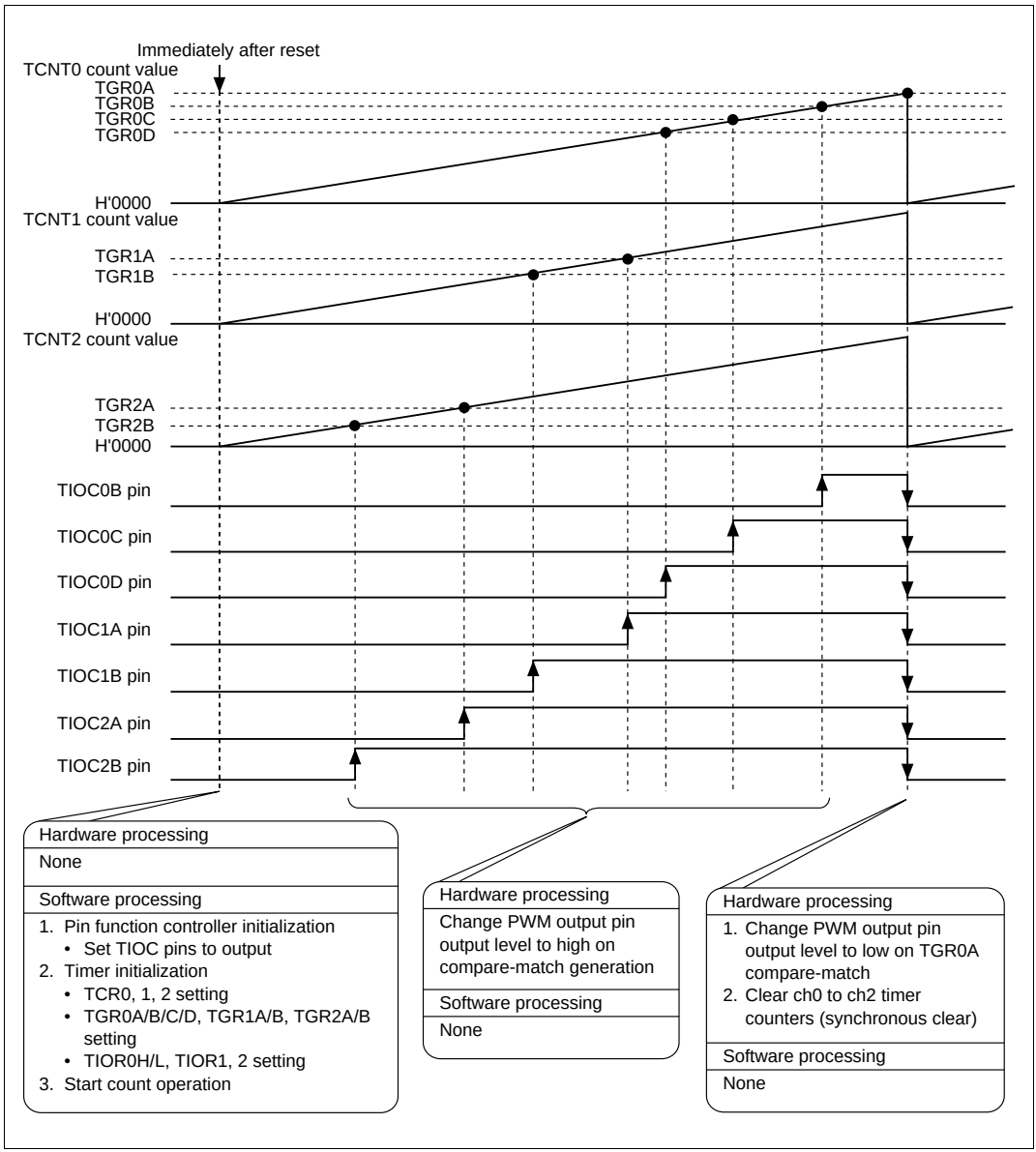


Figure 2-4-3 Principles of PWM Output (7-Phase) Operation Used for Sawtooth Waveform Generation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	pwm_2	PFC and PWM output setting

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
pul_cyc0a	Setting of timer value corresponding to pulse cycle Pulse cycle is calculated from following formula: Pulse cycle (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	1 word	Main routine	Input
pul_duty0b pul_duty0c pul_duty0d pul_duty1b pul_duty2a pul_duty2b	Setting of timing for change of waveform output from TIOC pin			

3. Internal Registers Used

Register Name	Function Assignment	Module
PFCE.PECR2	Sets TIOC0B/C/D, TIOC1A/B, TIOC2A/B as output pins	Main routine
TMRSH.TSTR	Timer count start execution	
TMRSH.TSYR	Sets synchronous operation for timer counters 0 and 1	
T0.TCR0 T1.TCR1 T2.TCR2	Clear timer counter clear source on TGR0A compare-match Select ø as input clock	
T0.TGR0A	PWM cycle setting	
T0.TGR0B	Setting of timer counter value that causes high-level output from TIOC0B	
T0.TGR0C	Setting of timer counter value that causes high-level output from TIOC0C	
T0.TGR0D	Setting of timer counter value that causes high-level output from TIOC0D	
T1.TGR1A	Setting of timer counter value that causes high-level output from TIOC1A	
T1.TGR1B	Setting of timer counter value that causes high-level output from TIOC1B	
T2.TGR2A	Setting of timer counter value that causes high-level output from TIOC2A	
T1.TGR2B	Setting of timer counter value that causes high-level output from TIOC2B	
T0.TIOR0H	Sets 0 initial output and 0 output on output compare for TGR0A, and 0 initial output and 1 output on output compare for TGR0B	
T0.TIOR0L	Sets 0 initial output and 1 output on output compare for TGR0C, and 0 initial output and 1 output on output compare for TGR0D	
T1.TIOR1	Sets 0 initial output and 1 output on output compare for TGR1A, and 0 initial output and 1 output on output compare for TGR1B	
T1.TIOR2	Sets 0 initial output and 1 output on output compare for TGR1A, and 0 initial output and 1 output on output compare for TGR1B	
T0.TMDR0 T1.TMDR1 T2.TMDR2	Set operating mode for each channel to PWM mode 2	

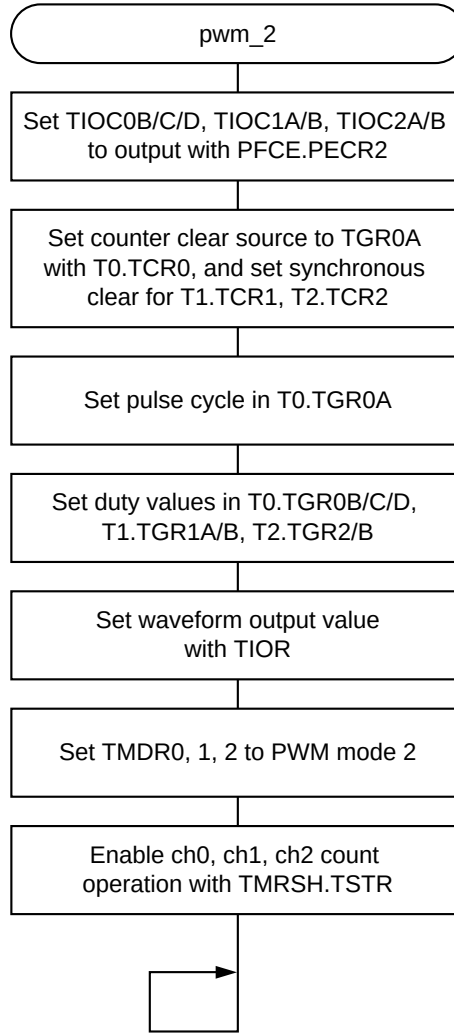
4. RAM Used

This application example does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

Main routine



Program List

```
/*---------------------------------------------------------------------*/
/*                            INCLUDE FILE                             */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*---------------------------------------------------------------------*/
/*                             PROTOTYPE                               */
/*---------------------------------------------------------------------*/
void pwm_2(void);
/*---------------------------------------------------------------------*/
/*                          RAM ALLOCATION                             */
/*---------------------------------------------------------------------*/
#define pul_cyc0      (*(unsigned short *)0xffffe800)
#define pul_duty0b    (*(unsigned short *)0xffffe802)
#define pul_duty0c    (*(unsigned short *)0xffffe804)
#define pul_duty0d    (*(unsigned short *)0xffffe806)
#define pul_duty1a    (*(unsigned short *)0xffffe808)
#define pul_duty1b    (*(unsigned short *)0xffffe80a)
#define pul_duty2a    (*(unsigned short *)0xffffe80c)
#define pul_duty2b    (*(unsigned short *)0xffffe80e)
/*---------------------------------------------------------------------*/
/*                            MAIN PROGRAM                             */
/*---------------------------------------------------------------------*/
void pwm_2(void)
     {
     PFCE.PEIOR = 0x00fe;     /* TIOC0B/C/D,TIOC1A/B,TIOC2A/B output */
     PFCE.PECR2 = 0x5554;     /* TIOC0B/C/D,TIOC1A/B,TIOC2A/B output */

     T0.TCR0 = 0x20;          /* TGR0A compare/match clear */
     T0.TIOR0H = 0x20;        /* start '0' compare/match '1' output */
     T0.TIOR0L = 0x22;        /* start '0' compare/match '1' output */
     T0.TCNT0 = 0x0000;       /* timer counter '0' set */
     T0.TGR0A = pul_cyc0;     /* timer general register set */
     T0.TGR0B = pul_duty0b;
     T0.TGR0C = pul_duty0c;
     T0.TGR0D = pul_duty0d;
     T0.TMDR0 = 0xc3;         /* pwm mode 2 */

     T1.TCR1 = 0x60;          /* TGR0A compare/match clear */
     T1.TIOR1 = 0x22;         /* start '0' compare/match '1' output */
     T1.TCNT1 = 0x0000;       /* timer counter '0' set */
     T1.TGR1A = pul_duty1a;   /* timer general register set *1
     T1.TGR1B = pul_duty1b;
     T1.TMDR1 = 0xc3;         /* pwm mode 2 *1

     T2.TCR2 = 0x60;          /* TGR0A compare/match clear */
     T2.TIOR2 = 0x22;         /* start '0' compare/match '1' output */
     T2.TCNT2 = 0x0000;       /* timer counter '0' set */
     T2.TGR2A = pul_duty2a;   /* timer general register set */
     T2.TGR2B = pul_duty2b;
     T2.TMDR2 = 0xc3;         /* pwm mode 2 */

     TMRSH.TSYR = 0x07;       /* ch0,1,2 synchronize */
     TMRSH.TSTR = 0x07;       /* timer start */
     while(1);
     }
```

2.5 Positive-Phase and Opposite-Phase PWM 3-Phase Output — MTU (Reset-Synchronized PWM Mode)

Specifications

1. The pulse high-level width is varied to generate positive-phase and opposite-phase 3-phase output of variable-duty pulses (duty pulses) as shown in figure 2-5-1.

2. At 28.7 MHz operation, any output pulse cycle from 34.8 ns to 2.28 ms can be set.

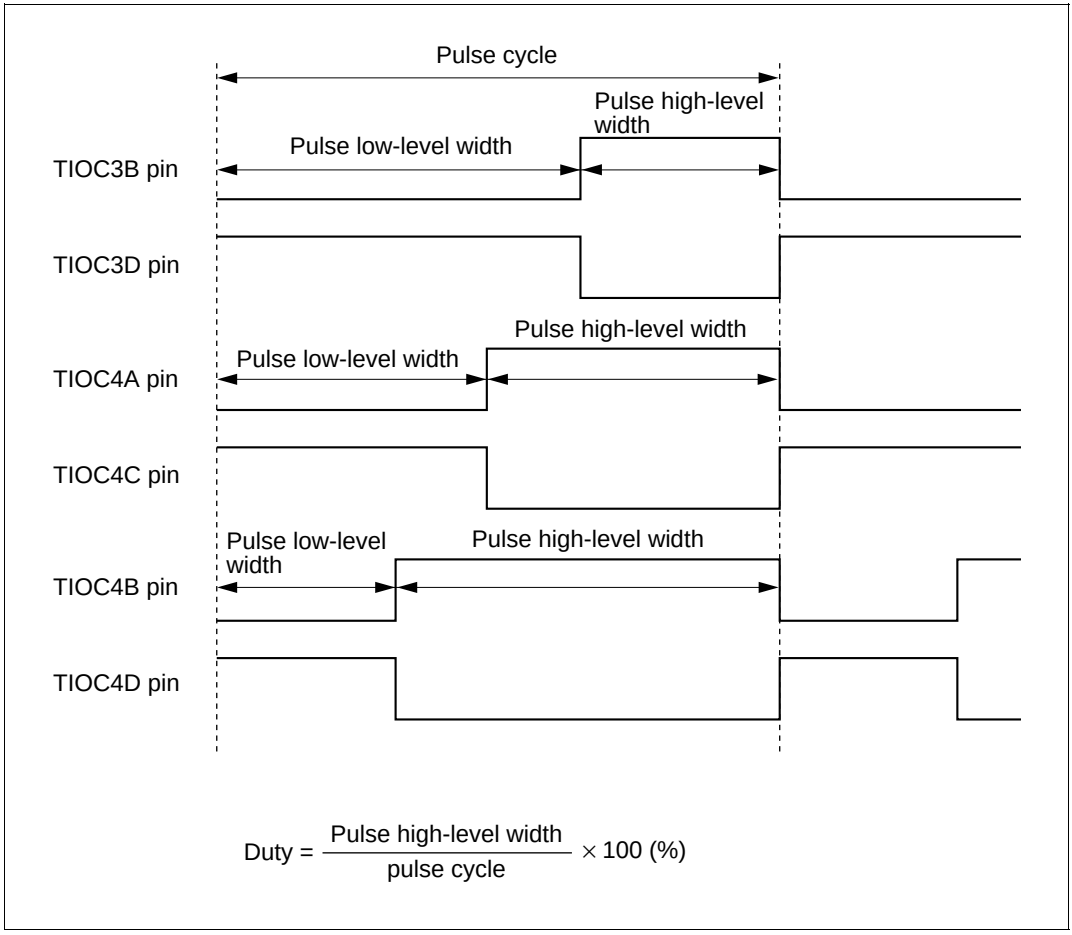


Figure 2-5-1 Example of Positive-Phase and Opposite-Phase PWM 3-Phase Output Waveform

Functions Used

1. In this sample task, 3-phase output is performed of PWM waveforms with a positive-phase/opposite-phase relationship and a common turning point, using MTU ch3 and ch4 in combination.

   In reset-synchronized PWM mode, PWM output is generated using the TGRA and TGRC registers as a pair, and the TGRB and TGRD registers as a pair.

   a. Figure 2-5-2 shows the MTU block diagram for this sample task.

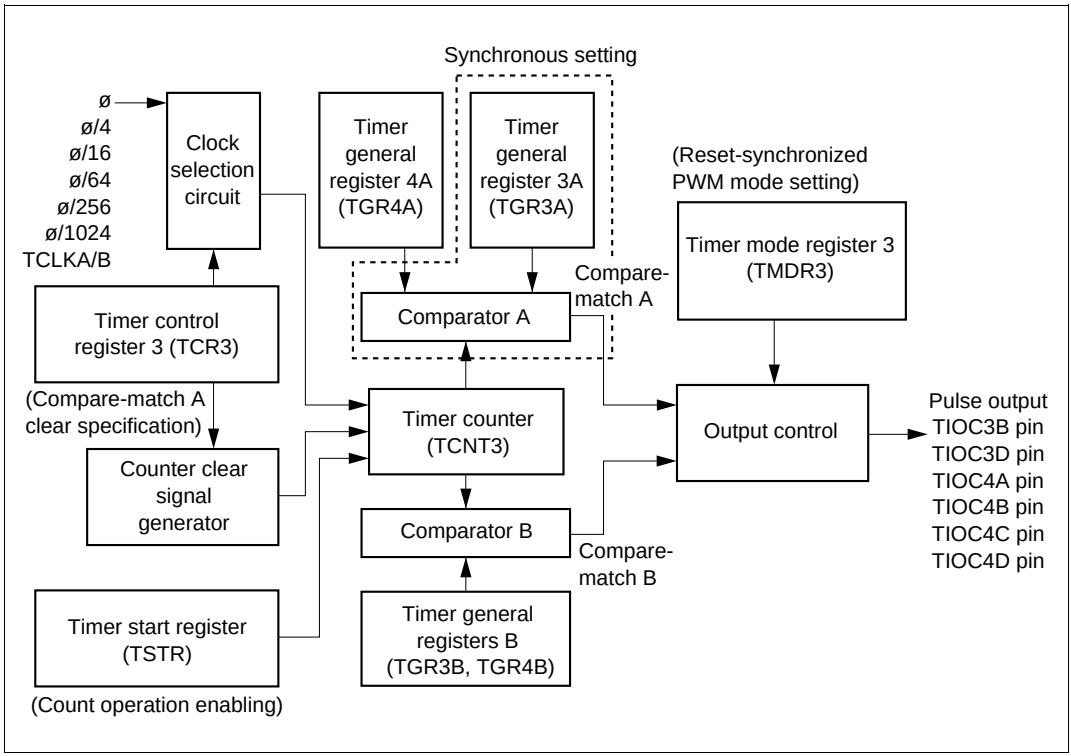


Figure 2-5-2 MTU/ch3, ch4 Block Diagram

2. Table 2-5-1 shows the function assignments for this task. MTU functions are assigned as shown in this table to perform PWM output.

Table 2-5-1 Function Assignment

Pin/Register Name	Function Assignment
TIOC3B	PWM output 1
TIOC3D	PWM output 1 opposite-phase waveform
TIOC4A	PWM output 2
TIOC4B	PWM output 3
TIOC4C	PWM output 2 opposite-phase waveform
TIOC4D	PWM output 3 opposite-phase waveform
TCR3	Selects ch3 timer counter clear source and input clock
TMDR3	Specifies ch3 operating in reset-synchronized PWM mode
TGR3A	PWM cycle setting
TGR3B TGR4A TGR4B	Duty value settings

Operation

Figure 2-5-3 shows the principles of the operation. 6-phase PWM waveforms are output from the PWM output pins (TIOC3B/D, TIOC4A/B/C/D) by means of SH7040 hardware and software processing as shown in the figure.

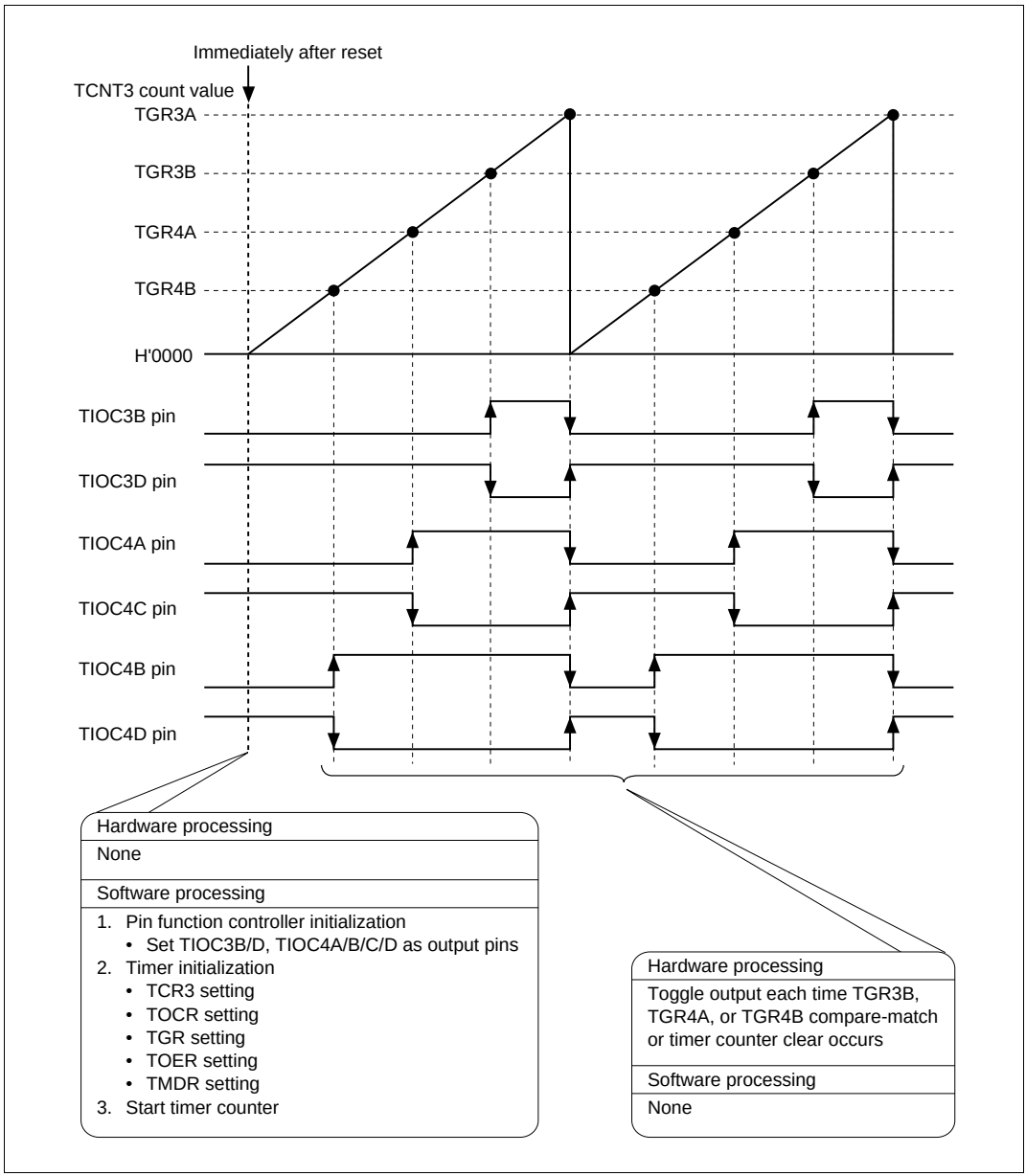


Figure 2-5-3 Principles of Reset-Synchronized PWM Waveform Output Operation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	rst_pwm	PFC and PWM output setting

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
pul_cyc1	Setting of timer value corresponding to pulse cycle Pulse cycle is calculated from following formula: Pulse cycle (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	1 word	Main routine	Input
pul_duty3b pul_duty4a pul_duty4b	Setting of timing for change of waveform output from TIOC pin			

3. Internal Registers Used

Register Name	Function Assignment	Module
RFCE.PECR1	Sets TIOC3B/D, TIOC4A/B/C/D as output pins	Main routine
TMRSH.TSTR	ch3 timer count start execution	
T3.TCR3	Clears timer counter clear source on TGR3A compare-match Selects ø as input clock	
T3.TOCR	Enabling of toggle output synchronized with PWM cycle, and positive-phase and opposite-phase output level setting	
T31.TGR3A	PWM cycle setting	
T31.TGR3B	Setting of timer counter value that causes toggle output from TIOC3B/D	
T31.TGR4A	Setting of timer counter value that causes high-level output from TIOC4A/C	
T31.TGR4B	Setting of timer counter value that causes high-level output from TIOC4B/D	
T3.TOER	Reset-synchronized PWM output enable setting	
T3.TMDR3	Reset-synchronized PWM mode setting	

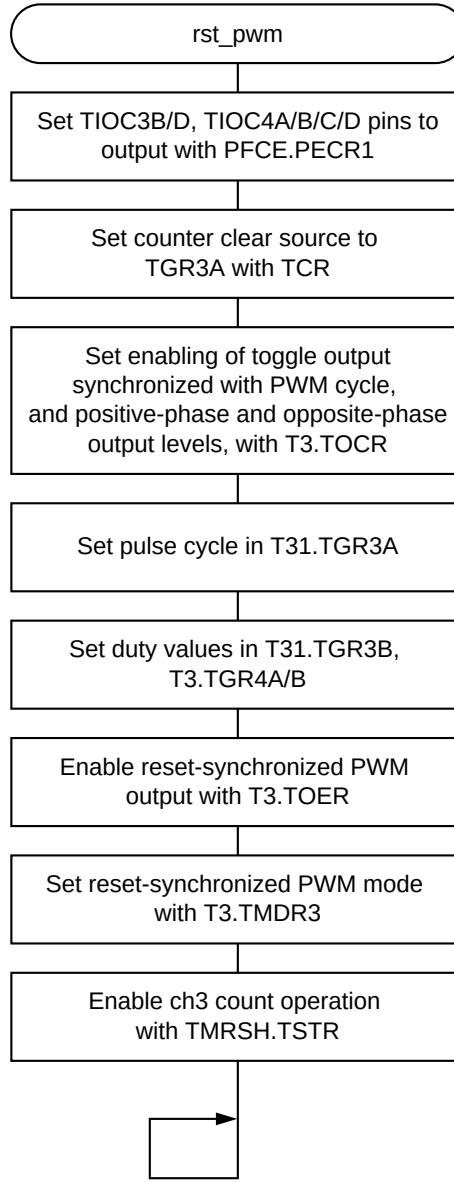
4. RAM Used

This application example does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



Program List

```
/*---------------------------------------------------------------------*/
/*                            INCLUDE FILE                             */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*---------------------------------------------------------------------*/
/*                             PROTOTYPE                               */
/*---------------------------------------------------------------------*/
void rst_pwm(void);
/*---------------------------------------------------------------------*/
/*                           RAM ALLOCATION                            */
/*---------------------------------------------------------------------*/
#define pul_cyc1      (*(unsigned short *)0xffffe800)
#define pul_duty3b    (*(unsigned short *)0xffffe802)
#define pul_duty4a    (*(unsigned short *)0xffffe804)
#define pul_duty4b    (*(unsigned short *)0xffffe806)
/*---------------------------------------------------------------------*/
/*                            MAIN PROGRAM                             */
/*---------------------------------------------------------------------*/
void rst_pwm(void)
     {
     PFCE.PEIOR = 0xfa00;     /* TIOC3B/D,TIOC4A/B/C/D output */
     PFCE.PECR1 = 0x5545;     /* TIOC3B/D,TIOC4A/B/C/D output */

     T3.TCR3 = 0x20;          /* TGR3A compare/match/clear */
     T31.TCNT3 = 0x0000;      /* timer counter '0' set */
     T31.TGR3A =pul_cyc1;     /* timer general register set */
     T31.TGR3B =pul_duty3b;

     T31.TCNT4 = 0x0000;      /* timer counter '0' set */
     T31.TGR4A = pul_duty4a;  /* timer general register set */
     T31.TGR4B = pul_duty4b;

     T3.TOER = 0xff;          /* timer output enable register */
     T3.TOCR = 0x43;          /* timer output control register */
     T3.TMDR3 = 0xc8;         /* reset-synchronized pwm mode */
     TMRSH.TSTR = 0x40;       /* timer start */
     while(1);
     }
```

Specifications

1. 3-phase output of PWM waveforms with non-overlapping of the positive phase and opposite phase is performed as shown in figure 2-6-1.

2. The duty can be varied arbitrarily from 0% to 100% by a setting in RAM.

$$\text{Duty} = \frac{\text{High-level pulse width}}{\text{pulse cycle}} \times 100\ (\%)$$

3. Toggle waveform output is performed in synchronization with the cycle.

4. At 28.7 MHz operation, any output pulse cycle from 69.6 ns to 2.28 ms can be set.

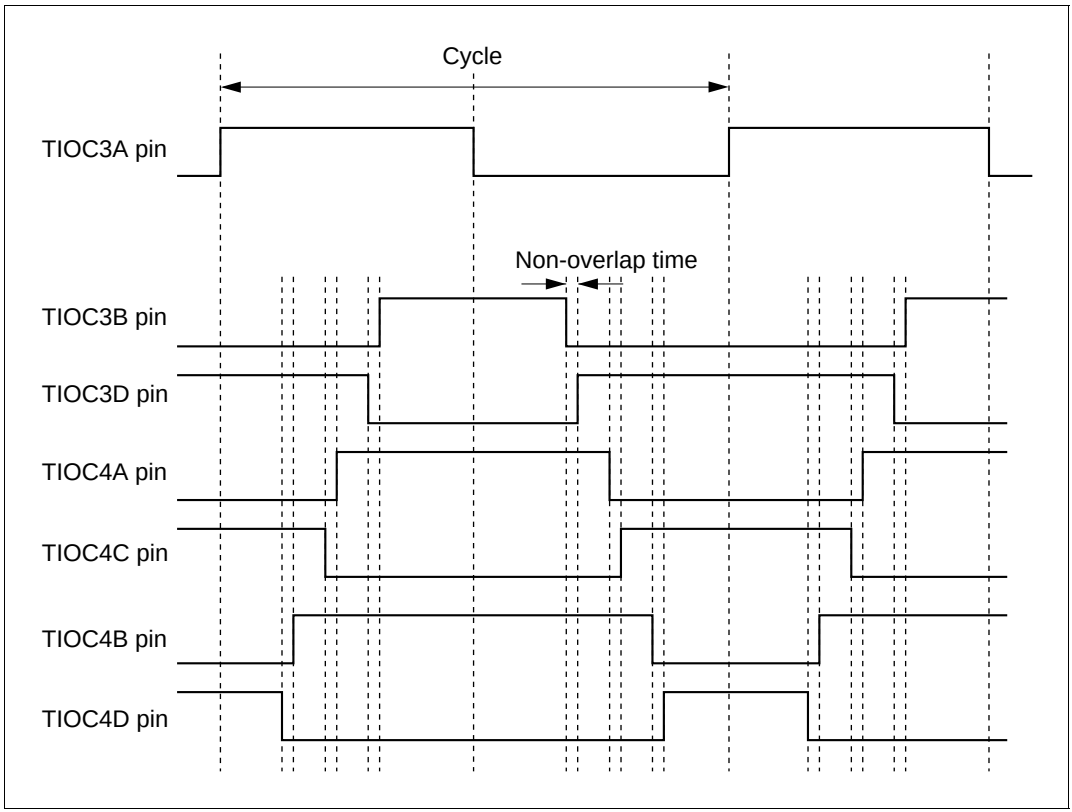


Figure 2-6-1 Complementary PWM 3-Phase Output Waveform

Functions Used

1. In this sample task, 3-phase output is performed of PWM waveforms with non-overlapping of the positive phase and opposite phase, using MTU ch3 and ch4 in combination. Toggle waveform output is also performed, synchronized with the PWM waveform cycle.

   a. Figure 2-6-2 shows the MTU/ch3, ch4 block diagram for this sample task. This task uses the following functions:

   - A function to generate 3-phase PWM waveform output with non-overlapping of positive and opposite phases (complementary PWM mode)
   - A function to transfer buffer register (TGR3C/D, TGR4C/D) contents to a compare register (TGR3A/B, TGR4A/B) when a compare-match occurs
   - A function to output a toggle waveform synchronized with the PWM waveform cycle

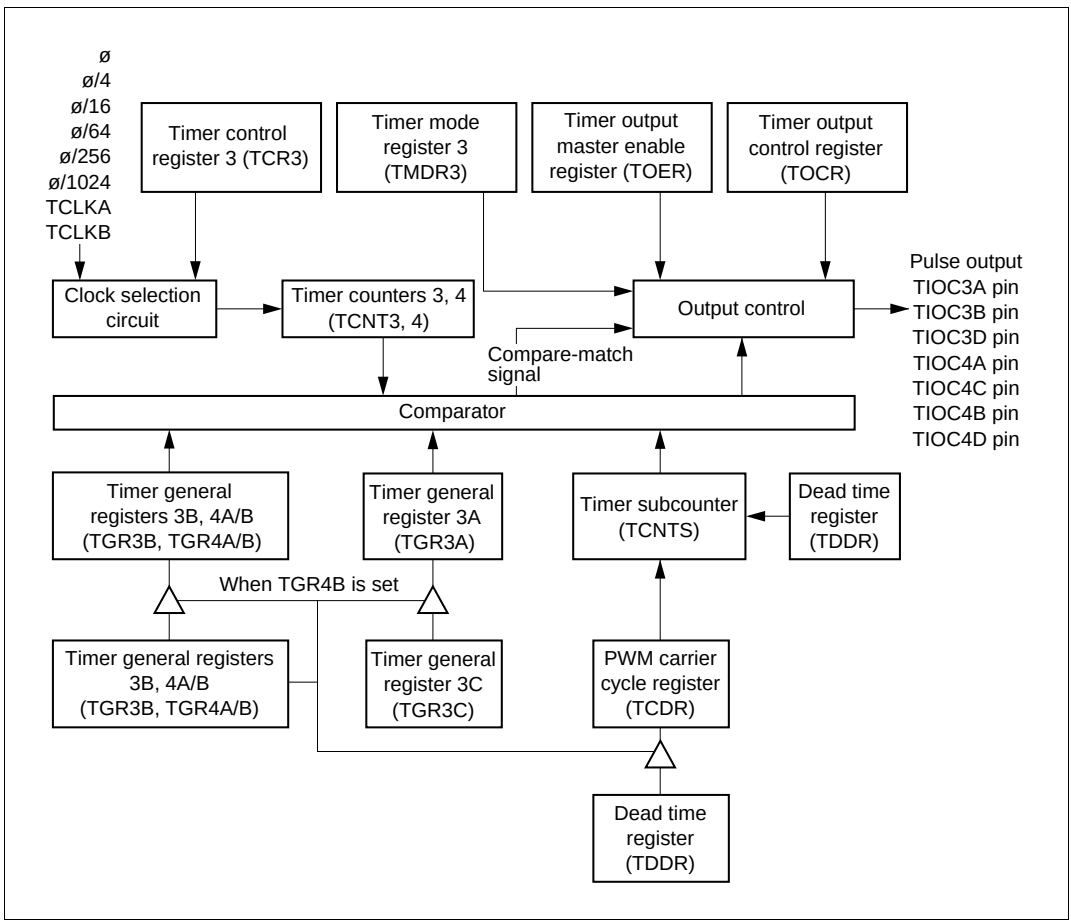


Figure 2-6-2 MTU/ch3, ch4 Block Diagram

2. Table 2-6-1 shows the function assignments for this task. MTU functions are assigned as

shown in this table to perform PWM pulse output.

Table 2-6-1 Function Assignment

Pin/Register Name	Function Assignment
TIOC3A	Toggle output synchronized with PWM
TIOC3C	PWM output 1
TIOC3D	Opposite-phase waveform in non-overlapping relationship with PWM output 1
TIOC4A	PWM output 2
TIOC4B	PWM output 3
TIOC4C	Opposite-phase waveform in non-overlapping relationship with PWM output 2
TIOC4D	Opposite-phase waveform in non-overlapping relationship with PWM output 3
TOCR	Enabling/disabling of toggle output synchronized with PWM cycle
TOER	Enabling/disabling of complementary PWM output pin signal output
TCR3	Selects ch3 timer counter clear source and input clock
TMDR3	Specifies ch3, ch4 operating in complementary PWM mode
TGR3A	Setting of 1/2 PWM cycle + dead time value
TGR3C	TGR3A buffer register
TGR3B	Output pulse turning point setting (compare register)
TGR4A	
TGR4B	
TGR4C	TGR4A buffer register
TGR4D	TGR4B buffer register
TDDR	Dead time setting
TCDR	1/2 cycle setting
TCBR	TCDR buffer register

Figure 2-6-3 shows the principles of the operation. Complementary PWM waveform output is performed by means of SH7040 hardware and software processing.

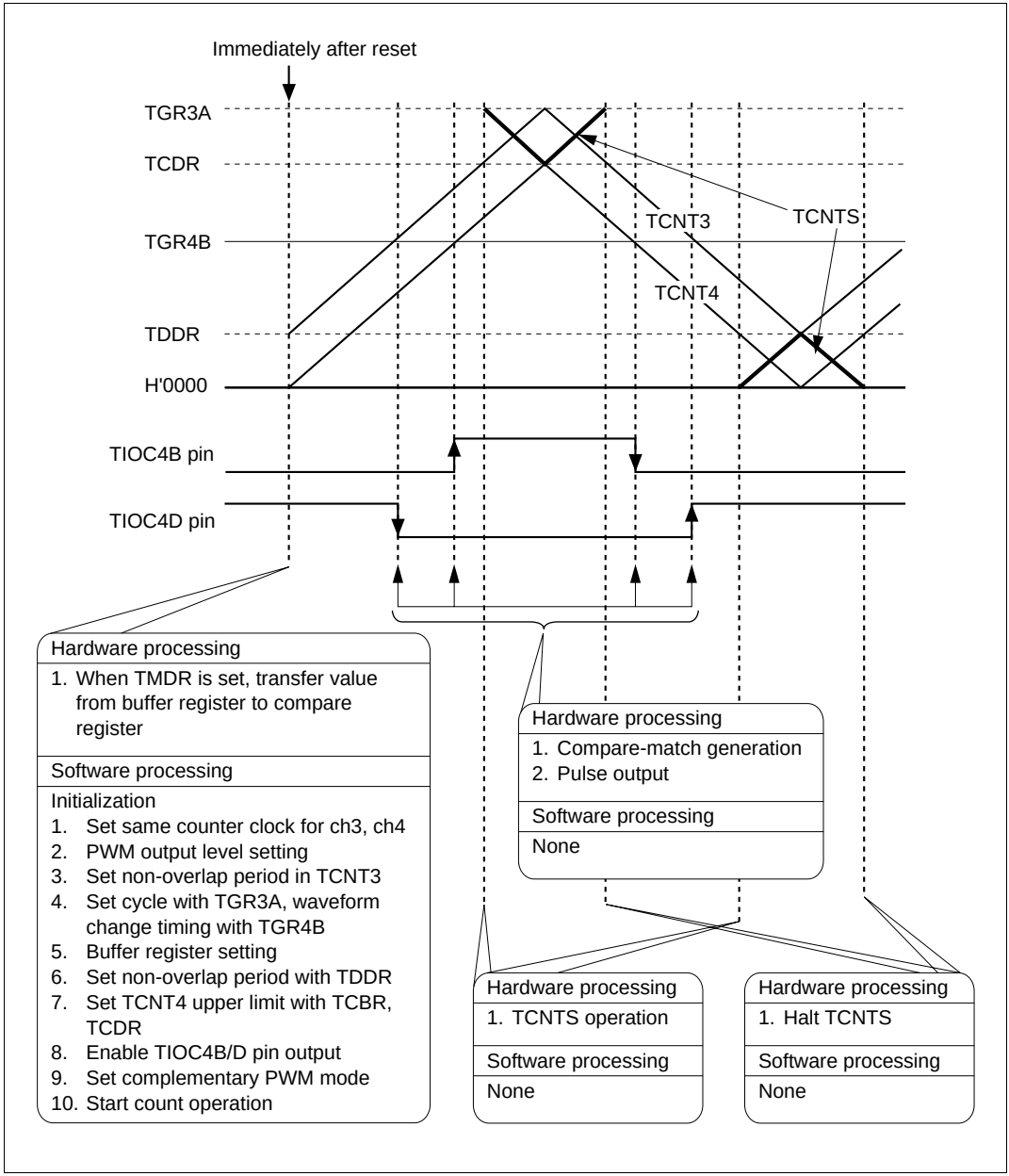


Figure 2-6-3 Principles of Complementary PWM Single-Phase Waveform Output Operation

Figure 2-6-4 shows the PWM waveform output method. When complementary PWM mode is set, the following rules apply to data transfer and compare operations.

Data Transfer

- In period Ta, data written in the buffer register is transferred to a temporary register (at the point at which data is set in TGR4D).
- In period Tb1, when transfer mode is set as transfer at the crest, data is not transferred from the buffer register to the temporary register. In period Tb2, the same operation is performed as in period Ta.

  Similarly, in the case of trough setting, data is not transferred in period Tb2.
- Data transfer to the buffer register can be performed as desired.

Compare

- In period Tb, the temporary register and compare register are compared with the TCNT3, TCNT4, and TCNTS counters to control the PWM waveform.
- In area (a), a compare-match of the pre-change data with (3) or (4) has priority.
- In area (b), a compare-match of the post-change data with (1) or (2) has priority.

However, generation of a compare-match at which the output waveform is set to the active level (compare-match (1) or (3)) occurs only after the generation of a compare at which the respective output waveform is set to the positive level (compare-match (4) or (2)).

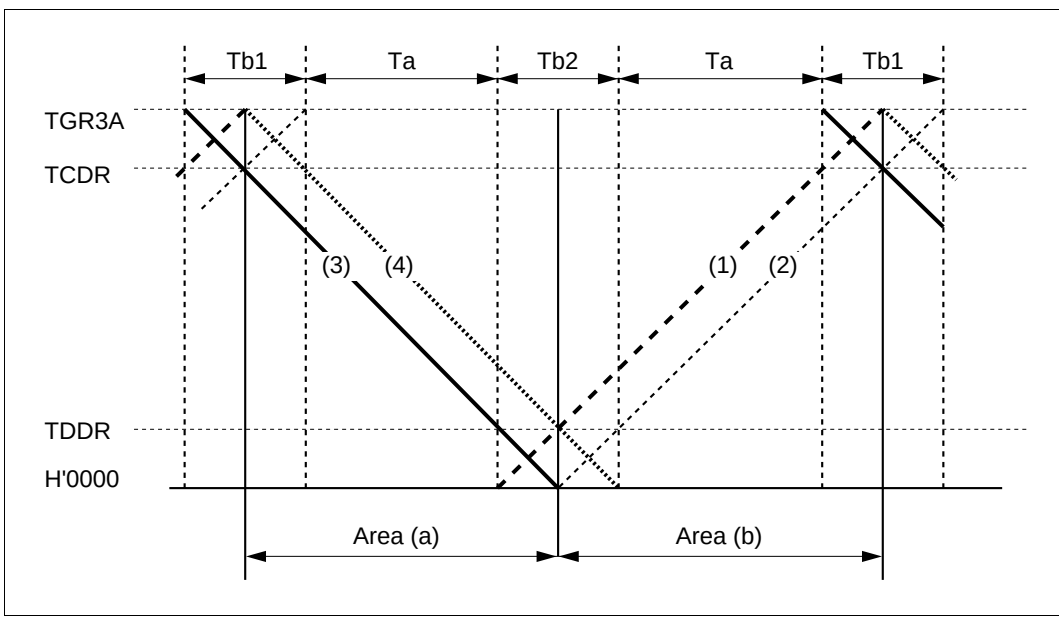


Figure 2-6-4 Principles of Operation of PWM Waveform Output Method

Figure 2-6-5 shows the principles of the operation. Complementary PWM waveform output is performed by means of SH7040 hardware and software processing. The transfer mode in which data is changed at the trough is selected.

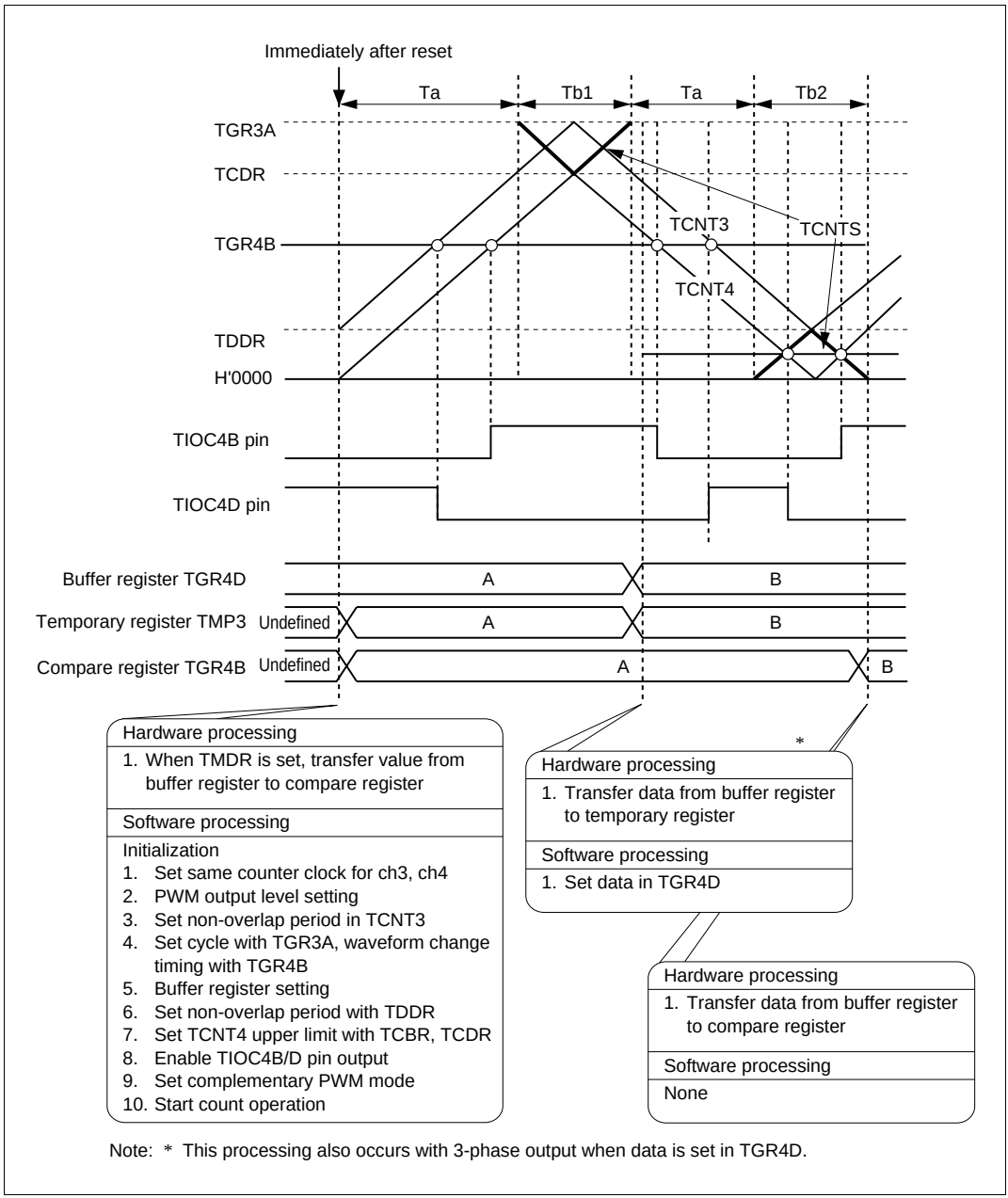


Figure 2-6-5 Principles of Complementary PWM Single-Phase Waveform Output Operation

Operation

Figure 2-6-6 shows the principles of the operation. 6-phase PWM output from the ch3 and ch4 PWM output pins (TIOC3B/D, TIOC4A/B/C/D) is performed by means of SH7040 hardware and software processing.

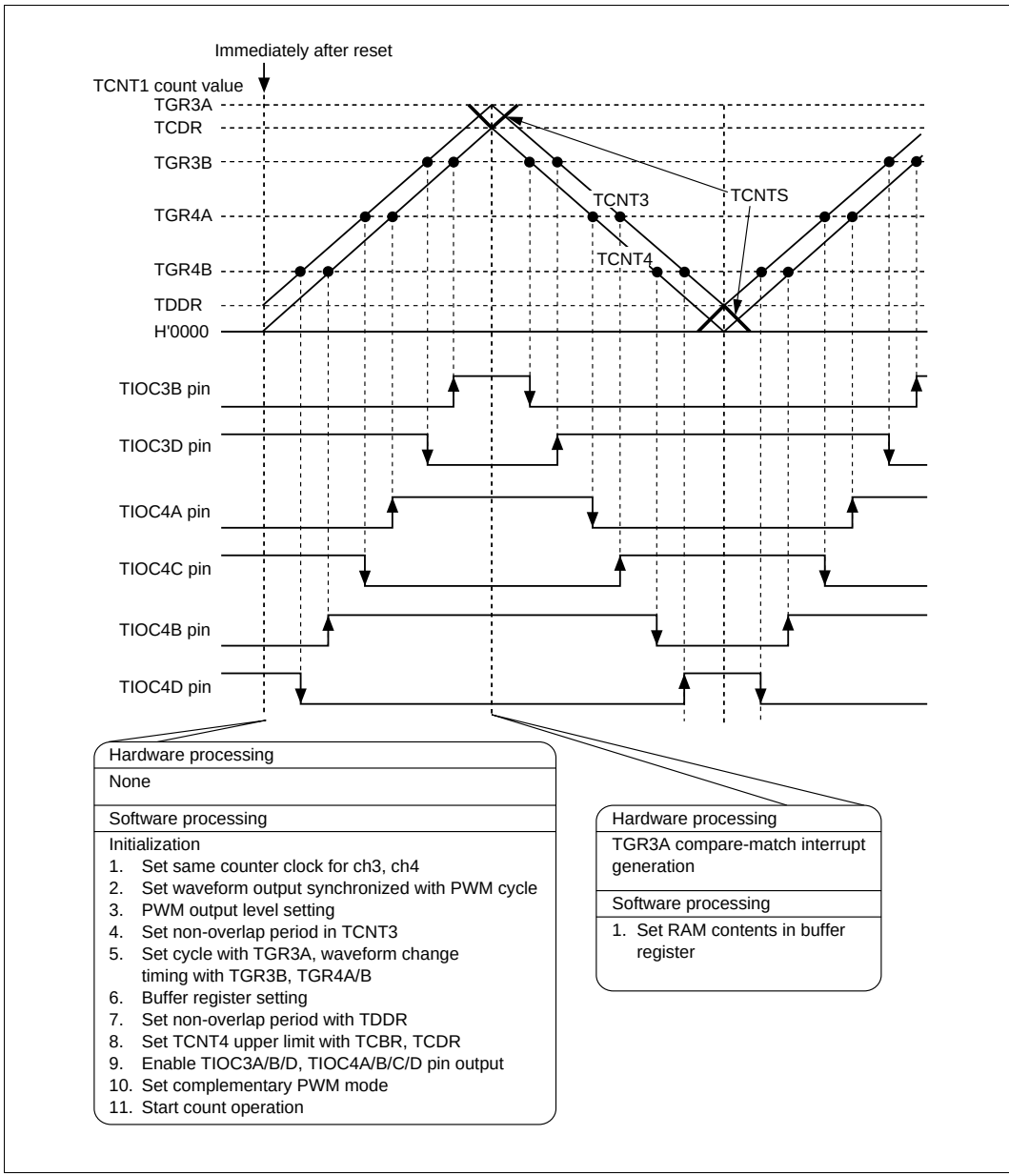


Figure 2-6-6 Principles of PWM Waveform Output Operation

Figure 2-6-7 shows the principles of the operation. Toggle output synchronized with the PWM cycle is performed by means of SH7040 hardware and software processing.

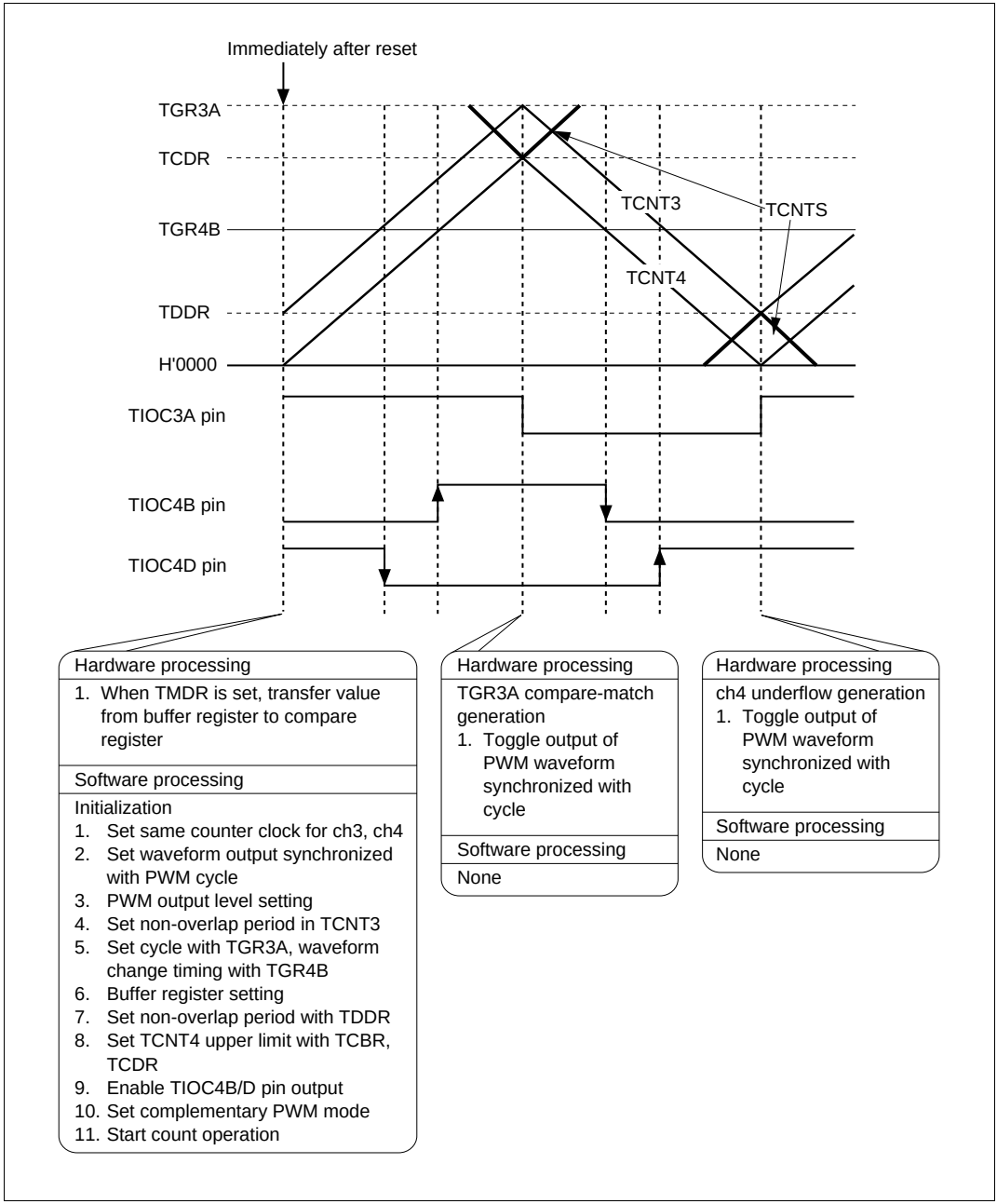


Figure 2-6-7 Principles of Toggle Waveform Output Operation Synchronized with PWM Cycle

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	comple	Complementary PWM output setting
Data setting	setdata	Sets waveform change timing in buffer register

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
pul_cyc1	Setting of 1/2 pulse cycle + dead time value Pulse cycle is calculated from following formula: Pulse cycle (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	1 word	Main routine	Input
pul_duty3d	Setting of timing for change of waveform output from TIOC pin			
pul_duty4c				
pul_duty4d				
c_cyc	PWM carrier cycle register value setting			
dead_time	Non-overlap period setting		Main routine Data setting	

3. Internal Registers Used

Register Name	Function Assignment	Module
PFCE.PEIOR	Sets TIOC3B/D, TIOC4A/B/C/D pins to output	Main routine
PFCE.PECR1	Sets TIOC3B/D, TIOC4A/B/C/D pins to MTU output	
TMRSH.TSTR	Timer count start execution	
T3.TCR3	Selects timer counter clear source and input clock	
T3.TIER3	Enables TGR3A interrupt	
T31.TGR3A	Setting of 1/2 carrier cycle + dead time register value	
T31.TGR3B	Setting of timer counter value that causes output from TIOC3B, TIOC3D	
T31.TGR3C	T31.TGR3A buffer register	Main routine Data setting
T31.TGR3D	T31.TGR3B buffer register	
T31.TCNT3	Dead time value setting	Main routine
T31.TGR4A	Setting of timer counter value that causes output from TIOC4A, TIOC4C	
T31.TGR4B	Setting of timer counter value that causes output from TIOC4B, TIOC4D	
T31.TGR4A	T31.TGR4A buffer register	Main routine Data setting
T31.TGR4B	T31.TGR4B buffer register	
T31.TDDR	Dead time value setting	Main routine
T31.TCDR	Setting of 1/2 cycle value	
T31.TCBR	T31.TCDR buffer register	Main routine Data setting
T3.TOCR	Enabling of toggle output synchronized with PWM cycle, and positive-phase and opposite-phase output level setting	Main routine
T3.TOER	Complementary PWM output enable setting	
T3.TMDR3	Set complementary PWM mode	
INTC.IPRE	Sets TGI0A compare-match interrupt priority level to 15	

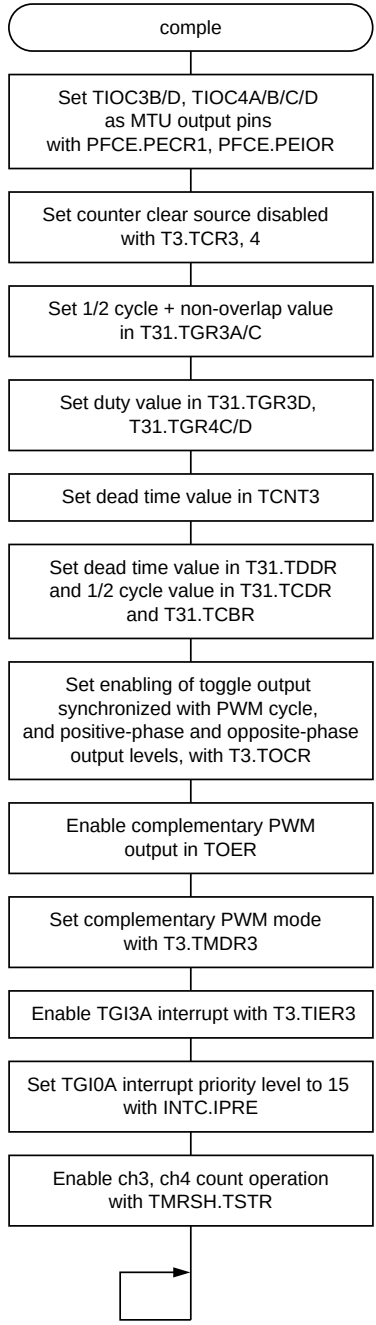
4. RAM Used

This application example does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



Program List

```
/*---------------------------------------------------------------------*/
/*                            INCLUDE FILE                             */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*---------------------------------------------------------------------*/
/*                             PROTOTYPE                               */
/*---------------------------------------------------------------------*/
void comple(void);
#pragma interrupt(setdata)
/*---------------------------------------------------------------------*/
/*                           RAM ALLOCATION                            */
/*---------------------------------------------------------------------*/
#define pul_cyc1        (*(unsigned short *)0xffffe800)
#define pul_duty3d      (*(unsigned short *)0xffffe802)
#define pul_duty4c      (*(unsigned short *)0xffffe804)
#define pul_duty4d      (*(unsigned short *)0xffffe806)
#define c_cyc           (*(unsigned short *)0xffffe808)
#define dead_time       (*(unsigned short *)0xffffe80a)
/*---------------------------------------------------------------------*/
/*                            MAIN PROGRAM                             */
/*---------------------------------------------------------------------*/
void comple(void)
     {
     PFCE.PEIOR = 0xfb00;     /* TIOC3B/D,TIOC4A/B/C/D output */
     PFCE.PECR1 = 0x5545;     /* TIOC3B/D,TIOC4A/B/C/D output */

     T3.TCR3 = 0x00;          /* not clear */
     T31.TGR3C = pul_cyc1;    /* TGR3A buffer register */
     T31.TGR3D = pul_duty3d;  /* TGR3D buffer register */
     T31.TCNT3 = dead_time;   /* dead time set */

     T3.TCR4 = 0x00;          /* don't clear */
     T31.TGR4C = pul_duty4c;  /* TGR4A buffer register */
     T31.TGR4D = pul_duty4d;  /* TGR4B buffer register */
     T31.TCNT4 = 0x0000;      /* timer '0' set */
     T31.TDDR = dead_time;    /* dead time register */
     T31.TCBR = c_cyc;        /* TCDR buffer register */

     T3.TOCR = 0x43;          /* timer output control register */
     T3.TOER = 0xff;          /* timer output enable register */
     T3.TMDR3 = 0xff;         /* coplementary-pwm mode */
     T3.TIER3 = 0x01;         /* timer interrupt enable register */
     INTC.IPRE = 0x00f0;      /* set initialize level=15 */
     set_imask(0x0);          /* set imask level=0 */
     TMRSH.TSTR = 0xc0;       /* timer start */
     while(1);
     }

void setdata()
{
    T31.TSR3 &= 0xfe;         /* clear flag */
    T31.TCBR = c_cyc;
    T31.TGR3C = pul_cyc1;
    T31.TGR3D = pul_duty3d;
    T31.TGR4C = pul_duty4c;
    T31.TGR4D = pul_duty4d;
}
```

Specifications

1. Two external clocks are input to ch1, and the counter is incremented or decremented according to the phase difference of the pulses, as shown in figure 2-7-1. The ch1 count value is measured in synchronization with the measurement times (measurement time 1/2) set for ch0, and the result is stored in RAM.

2. The initial value of the timer counter is H'0000, and counting can be performed from –2147483648 to 2147483647 using a software counter.

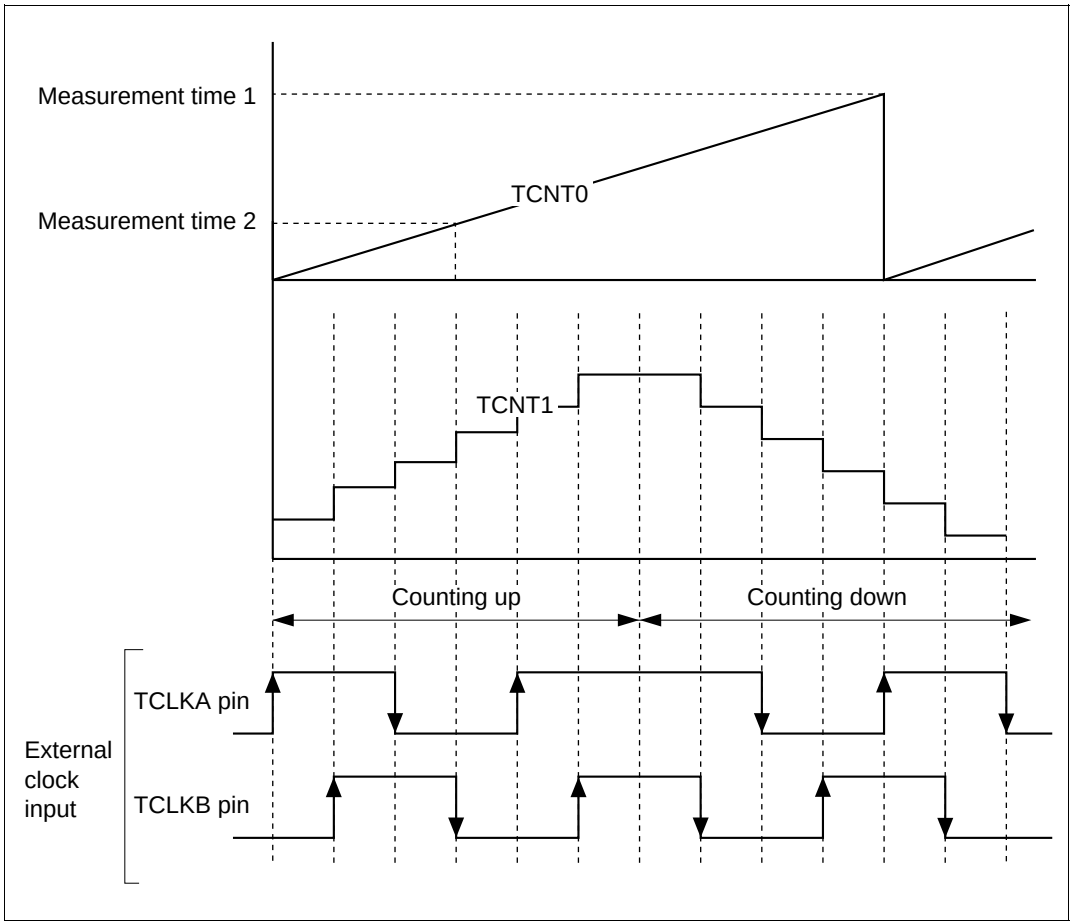


Figure 2-7-1 2-Phase Encoder Counter Latching

Functions Used

1. In this sample task, MTU ch1 is used as an up/down-counter, and the measurement times are set in TGR0A/B. The TCNT1 value in a control cycle is latched by ch1 input capture with TGR0A/B output compare as the trigger. The ch1 counter input clock width is latched using ch0 input capture.

   a. Figure 2-7-2 shows the ch0 block diagram. Ch0 outputs a ch1 input capture trigger each measurement interval, using the following functions. Ch1 measures the TCNT1 value on input of the input capture signal.

      - A function for automatically outputting pulses by hardware without software intervention (output compare)
      - A function for detecting pulse input edges and latching the timer value in an internal register (input capture)

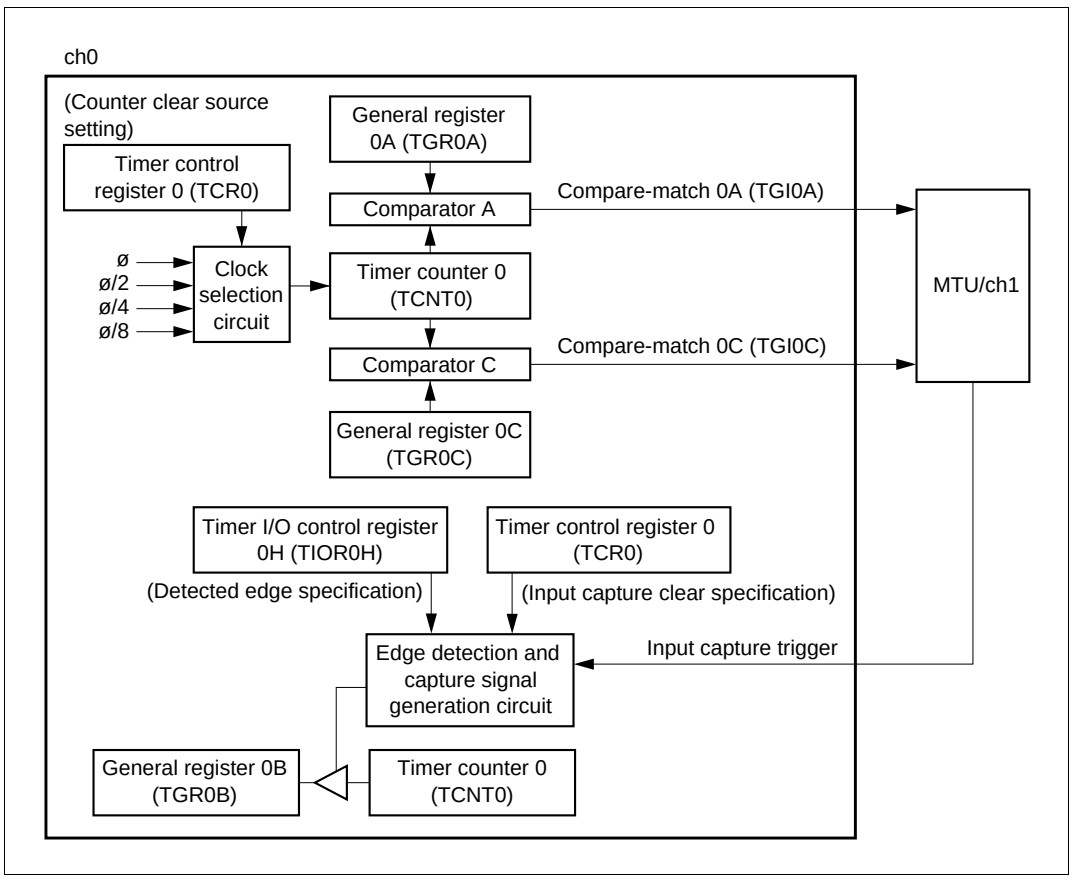


Figure 2-7-2 MTU/ch0 Block Diagram

b. Figure 2-7-3 shows the ch1 block diagram. In ch1 the timer counter is incremented/decremented using the following functions. The counter value at the time an input capture rising edge is detected is taken as the measurement result.

- A function for detecting the phase difference between two external clocks, and incrementing/decrementing the timer counter (phase counting mode)
- A function for detecting pulse input edges and latching the timer value at that time in an internal register (input capture)
- A function that initiates interrupt handling when input capture is generated
- A function that clears the timer counter when a pulse input edge is detected
- A function that initiates interrupt handling on detection of timer counter overflow or underflow

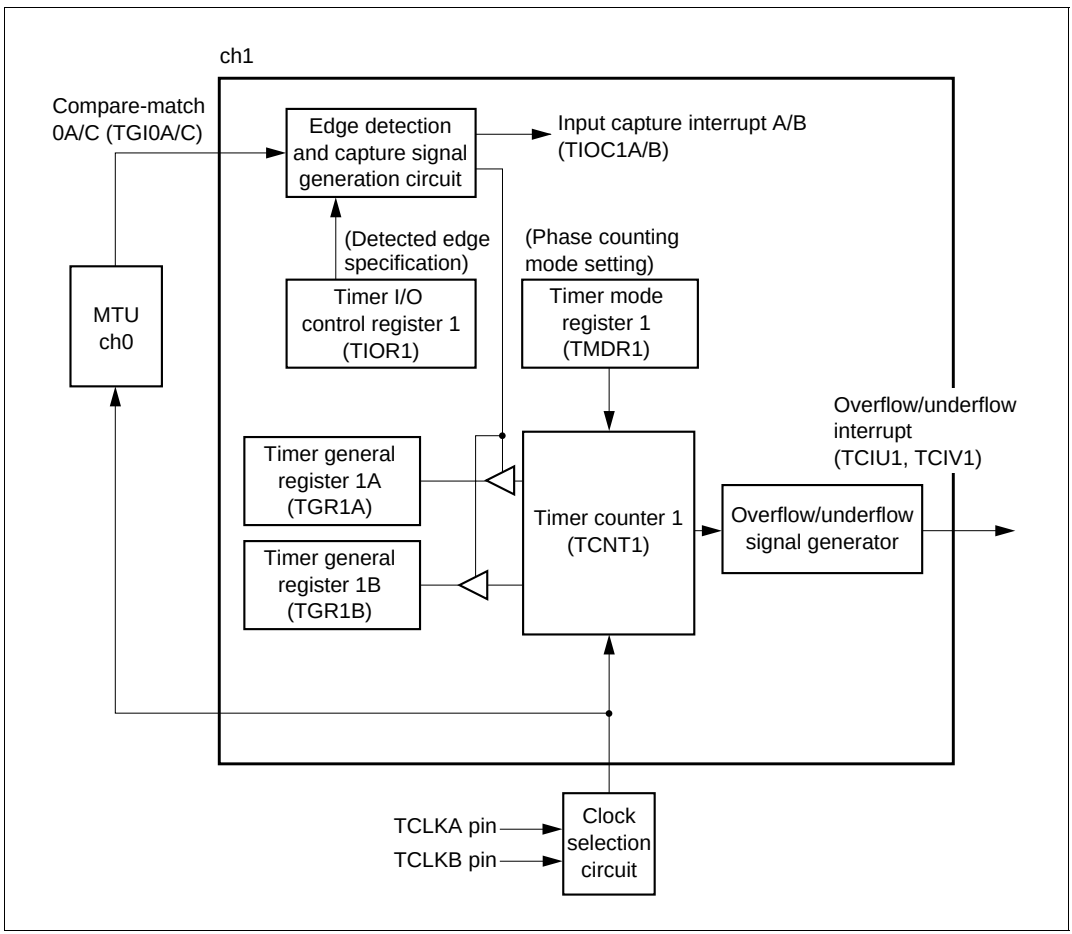


Figure 2-7-3 MTU/ch1 Block Diagram

2. Table 2-7-1 shows the function assignments for this sample task. MTU functions are assigned as shown in this table to detect the phase difference between two 2-phase encoder pulses, and incrementing/decrementing the counter.

Table 2-7-1 Function Assignment

Pin/Register Name	Function Assignment
TCLKA	External clock input pins
TCLKB	
TSTR	Enabling/disabling of ch0, ch1 timer counter operation
TCR0	Counter clock and counter clear source selection
TIOR0H	Sets TIOC0A to output compare. Sets TIOC0B to input capture on generation of ch0 output compare.
TIOR0L	Sets TIOC0C to output compare
TGR0A	Measurement time 1 setting
TGR0B	Holds input capture B count result
TGR0C	Set to 2 every measurement interval
TMDR1	Phase counting mode setting
TCR1	Counter clock and counter clear source selection
TIOR0	Sets TIOC0A/C to input capture on generation of ch1 output compare
TIER1	Enables TGI1A/B, TIOU1, TIOV1 interrupts
TGR1A	Holds input capture A count result
TGR1B	

Operation

Figure 2-7-4 shows the principles of the operation. The counter is incremented/decremented by means of SH7040 hardware and software processing.

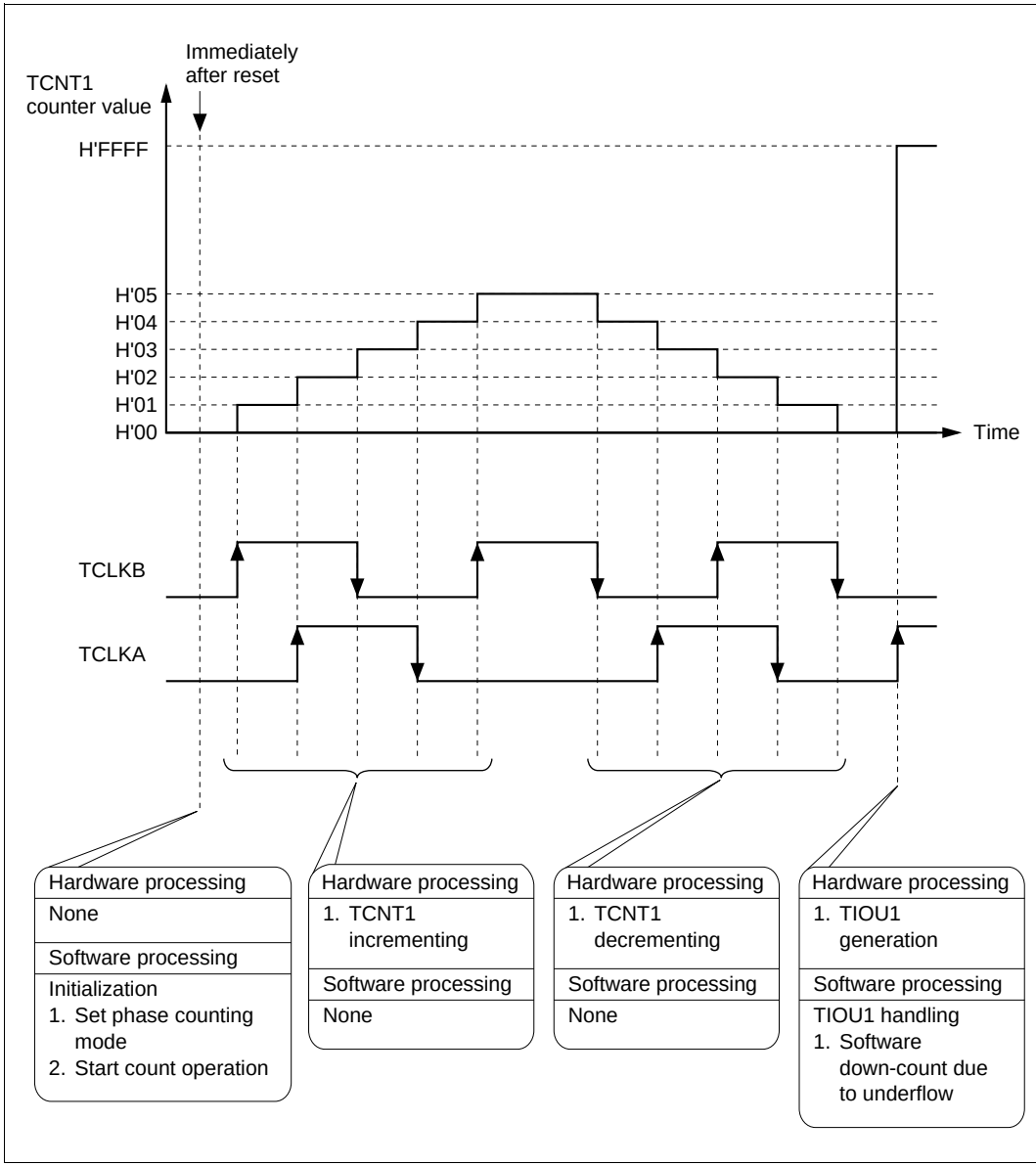


Figure 2-7-4 Principles of Phase Counting Mode Operation (1)

Interrupt handling is executed by means of SH7040 hardware and software processing in the case of counter overflow/underflow or external event occurrence, as shown in figure 2-7-5.

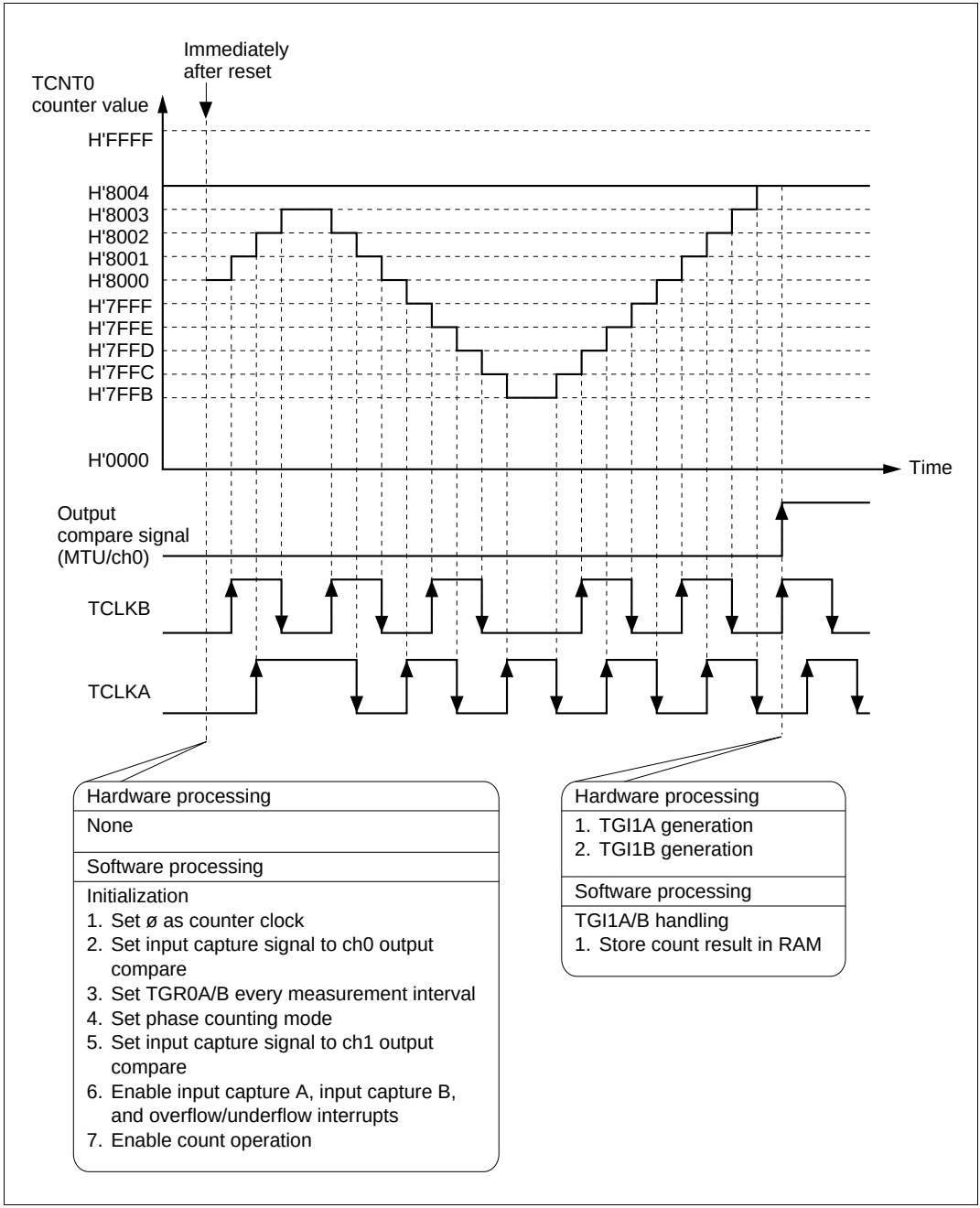


Figure 2-7-5 Principles of Phase Counting Mode Operation (2)

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	en2	Initialization of MTU, etc.
Counter value measurement 1	phacnt1	Initiated by TGI1A; sets up/down-count result in RAM from TGRA value Sets counter cycle result in RAM from TGRC value
Counter value measurement 2	phacnt2	Initiated by TGI1B; sets up/down-count result in RAM from TGRB value
Overflow	ovf1	Initiated by TIOV1; increments software counter
Underflow	unf1	Initiated by TIOU1; decrements software counter

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
msr_tim1 msr_tim2	Setting of timer value corresponding to counter measurement time Measurement time is found from following formula: Measurement time (ns) = Timer value × ø cycle (35 ns at 28.7 MHz operation)	Word	Main routine	Input
cnt–data1 cnt_data2	Setting of up/down-count result	Longword	Counter value measurement 1 Counter value measurement 2	Output
p_cycle	Setting of count cycle result	Word	Counter value measurement 1	

3. Internal Registers Used

Register Name	Function Assignment	Module
TMRSH.TSRT	Enabling/disabling of ch0/1 timer counter operation	Main routine
T0.TCR0	Counter clock and counter clear source selection	
T0.TIOR0H	Sets TIOC0A to output compare. Sets TIOC0B to input capture on generation of ch0 output compare.	
T0.TIOR0L	Sets TIOC0C to output compare	
T0.TGR0A	Measurement time 1 setting	
T0.TGR0B	Holds input capture B count result	Counter value measurement 1
T0.TGR0C	Set to 2 every measurement interval	Main routine
T1.TMDR1	Phase counting mode setting	
T1.TCR1	Counter clock and counter clear source selection	
T1.TIOR1	Sets TIOC0A/C to input capture on generation of ch1 output compare	
T1.TIER1	Enables TGI1A/B, TIOU1, TIOV1 interrupts	
T1.TGR1A	Holds input capture A count result	Counter value measurement 1
T1.TGR1B	Holds input capture B count result	Counter value measurement 2
T1.TSR1	TGI1A/B, TIOU1, TIOV1 generation status	Counter value measurement 1, 2 Overflow, Underflow

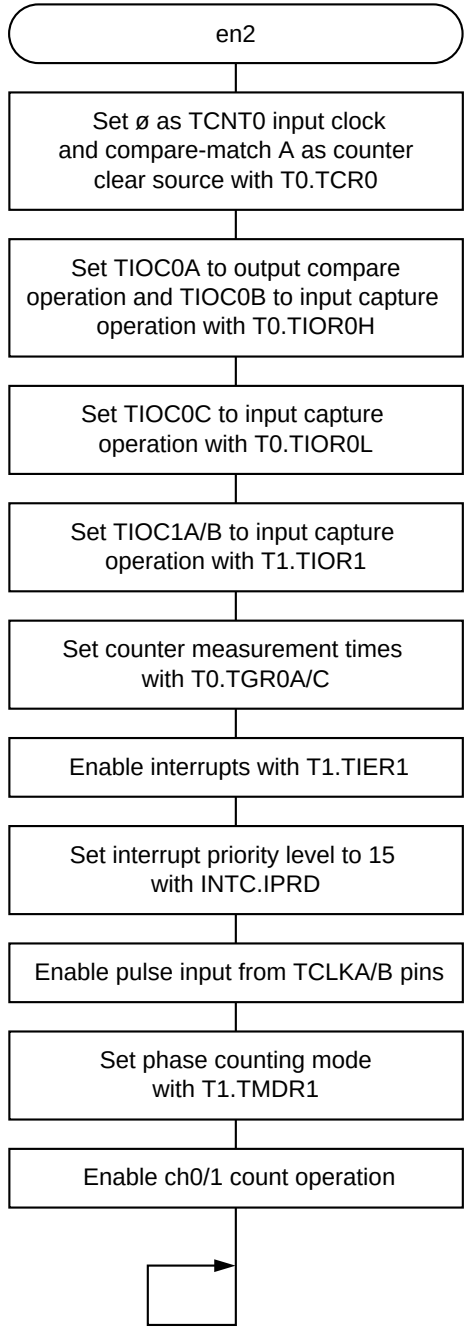
4. RAM Used

Module	Label	Function Assignment
Counter value measurement 1, 2	wrk	Used as work area when setting data
All modules	cnt	Software counter

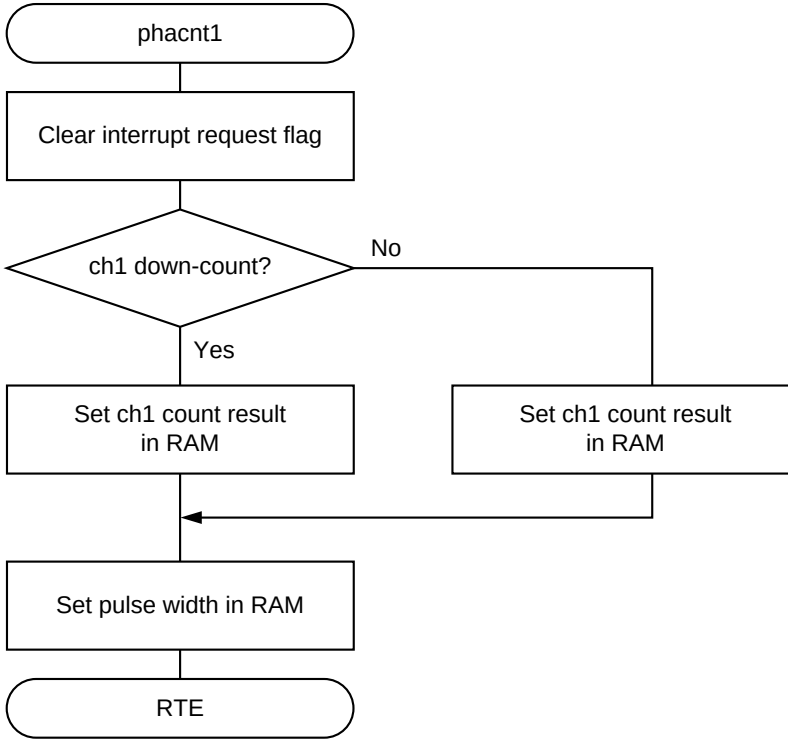
Note: The SH7040 header file name is used as the register label.

Flowchart

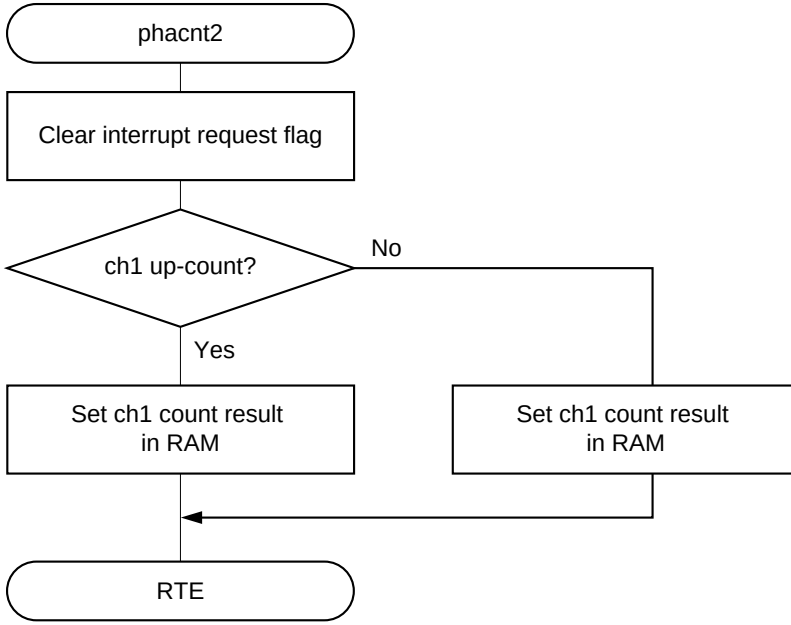
1. Main routine



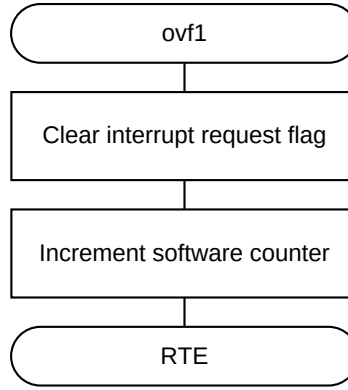
2. Counter value measurement 1



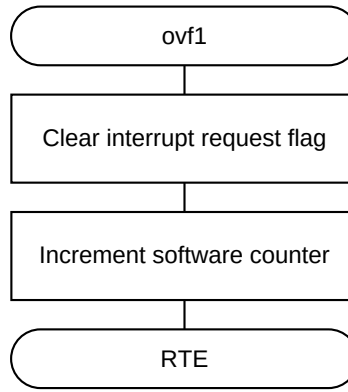
3. Counter value measurement 2



4. Overflow



5. Underflow



Program List

```
/*---------------------------------------------------------------------*/
/*                           INCLUDE FILE                              */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include "SH7040.H"
/*---------------------------------------------------------------------*/
/*                            PROTOTYPE                                */
/*---------------------------------------------------------------------*/
void en2(void);
#pragma interrupt(phacnt1,phacnt2,ovf1,unf1)
/*---------------------------------------------------------------------*/
/*                          RAM ALLOCATION                             */
/*---------------------------------------------------------------------*/
#define msr_tim1(*(unsigned short *)0xffffe800)
#define msr_tim2(*(unsigned short *)0xffffe802)
#define cnt_data2 (*(signed long *)0xffffe804)
#define cnt_data1 (*(signed long *)0xffffe808)
#define p_cycle (*(unsigned long *)0xffffe80c)
#define cnt     (*(signed long *)0xffffe810)
#define wrk     (*(unsigned short *)0xffffe814)
/*---------------------------------------------------------------------*/
/*                           MAIN PROGRAM                              */
/*---------------------------------------------------------------------*/
void en2(void)
{
    T0.TCR0 = 0xa0;        /* timer clear output compare TGRA0 */
    T0.TIOR0H = 0xf0;      /* output compare TIOC0A */
                                        /* input capture TIOC0B */
    T0.TIOR0L = 0x00;      /* output compare TIOC0C */
    T1.TIOR1 = 0xff;       /* input capture TIOC1A,B */
    T1.TIER1 = 0x33;       /* interrupt TIOC1A,TIOC1B,TCIU1,TCIV1 */
    T0.TGR0C = msr_tim2;   /* set position cycle */
    T0.TGR0A = msr_tim1;   /* set speed cycle */
    INTC.IPRD = 0x00ff;    /* set interrupt level=15 */
    set_imask(0x0);        /* set imask level=0 */
    PFCA.PACRL2 = 0x5000;  /* TIOCnx sellect */
    T1.TMDR1 = 0x04;       /* set phase counting mode1 */
    T0.TMDR0 = 0x20;       /* TGRB and TGRD buffer mode */
    TMRSH.TSTR = 0x03;     /* start timer0,1 */
    while(1);              /* loop */
}
```

```
void ovf1(void)
{
    T1.TSR1 &= 0xef;                                          /* clear flag */
    cnt++;                                                    /* count up */
}

void unf1(void)
{
    T1.TSR1 &= 0xdf;                                          /* clear flag */
    cut--;                                                    /* count down */
}

void phacnt1(void)
{
    T1.TSR1 &= 0xfe;                                           /* clear flag */
    wrk = T1.TGR1B;
    if(cnt<0)                                                  /* count<0 */
        cnt_data1 = (unsigned long)wrk-0x010000+cnt*0x010000; /* set sp */
    else
        cnt_data1 = (unsigned long)wrk+cnt*0x010000;           /* set sp */
        p_cycle = T0.TGR0D;                                    /* set width pulse */
}

void phacnt2(void)
{
    T1.TSR1 &= 0xfd;                                           /* clear flag */
     wrk = T1.TGR1A;
     if(cnt<0)
        cnt_data2 = (unsigned long)wrk-0x010000+cnt*0x010000; /* set po */
     else
        cnt_data2 = (unsigned long)wrk+cnt*0x010000;           /* set po */
}
```

Specifications

1. Waveform cutoff is performed by setting the timer output waveform to the high-impedance state in synchronization with the falling edge of an external signal, as shown in figure 2-8-1.

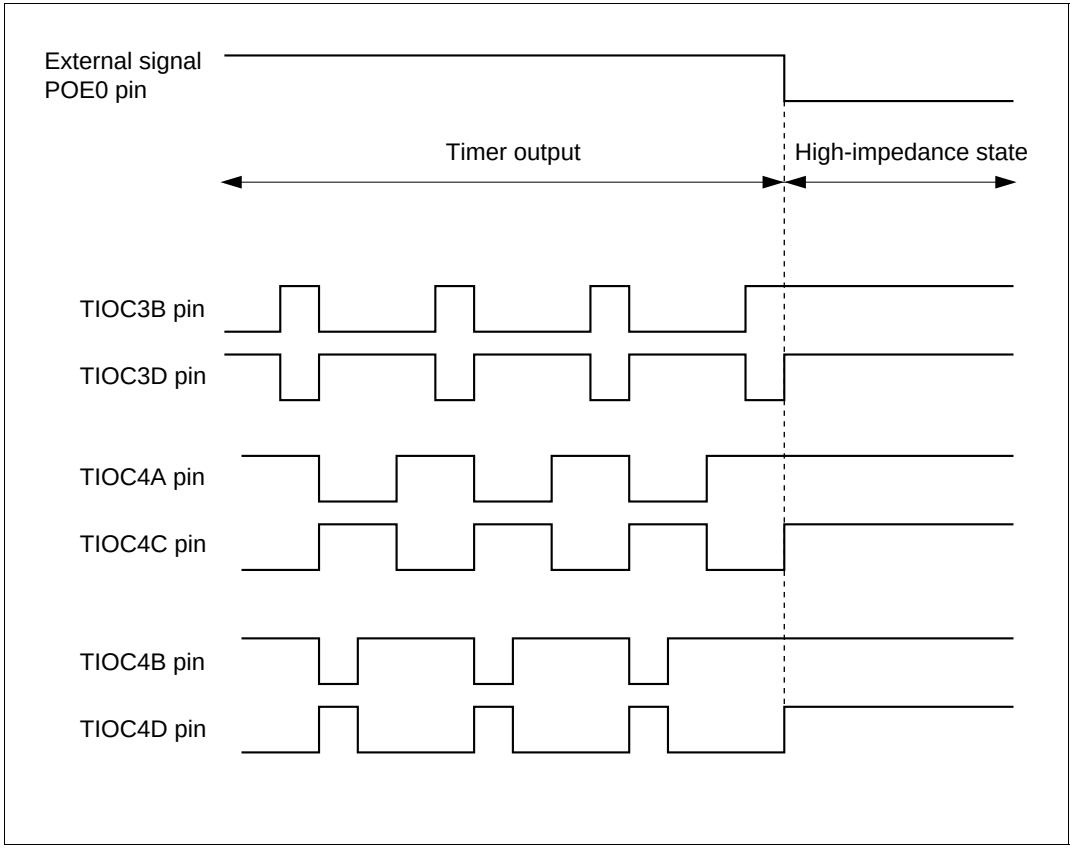


Figure 2-8-1 Example of Externally Triggered Timer Waveform Cutoff

Functions Used

1. In this sample task, waveform output cutoff is performed by setting waveforms output by MTU channels 3 and 4 (reset-synchronized PWM mode) to the high-impedance state in synchronization with the falling edge of an external signal.

   a. Figure 2-8-2 shows the MTU/ch3, ch4, POE block diagram.

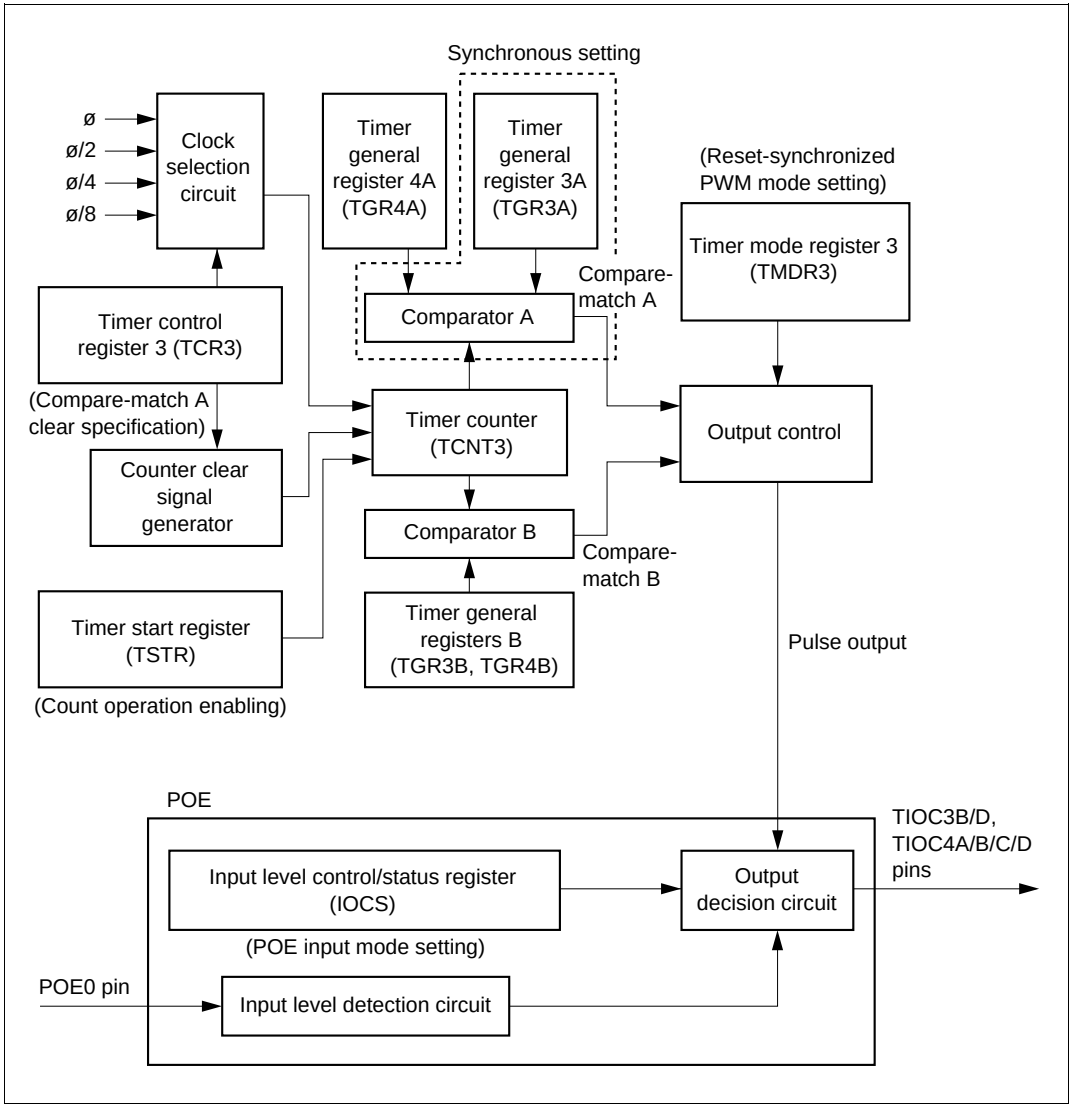


Figure 2-8-2 MTU/ch3, ch4, POE Block Diagram

2. Table 2-8-1 shows the function assignments for this task. MTU and POE functions are assigned as shown in this table to perform waveform cutoff.

Table 2-8-1 Function Assignment

Pin/Register Name	Function Assignment
TIOC3B	Pulse output pins
TIOC3D	
TIOC4A	
TIOC4B	
TIOC4C	
TIOC4D	
POE0	Cutoff external signal input pin
TSTR3	Enables/disables ch3 timer counter operation
TCR3	Selects ch3 timer counter clear source and input clock
TMDR3	Sets ch3, ch4 to reset-synchronized PWM mode
TGR3A	PWM cycle setting
TGR3B	Output waveform change timing setting
TGR3C	
TGR3D	
TGR4A	
TGR4B	
TOER	Enables/disables TIOC3B/D, TIOC4A/B/C/D pin timer output
ICSR	POE input mode selection

Operation

Figure 2-8-3 shows the principles of the operation. Waveform cutoff is performed automatically by hardware. (For the principles of reset-synchronized PWM operation, see section 2.5, Positive-Phase and Opposite-Phase PWM 3-Phase Output, in this Application Note.)

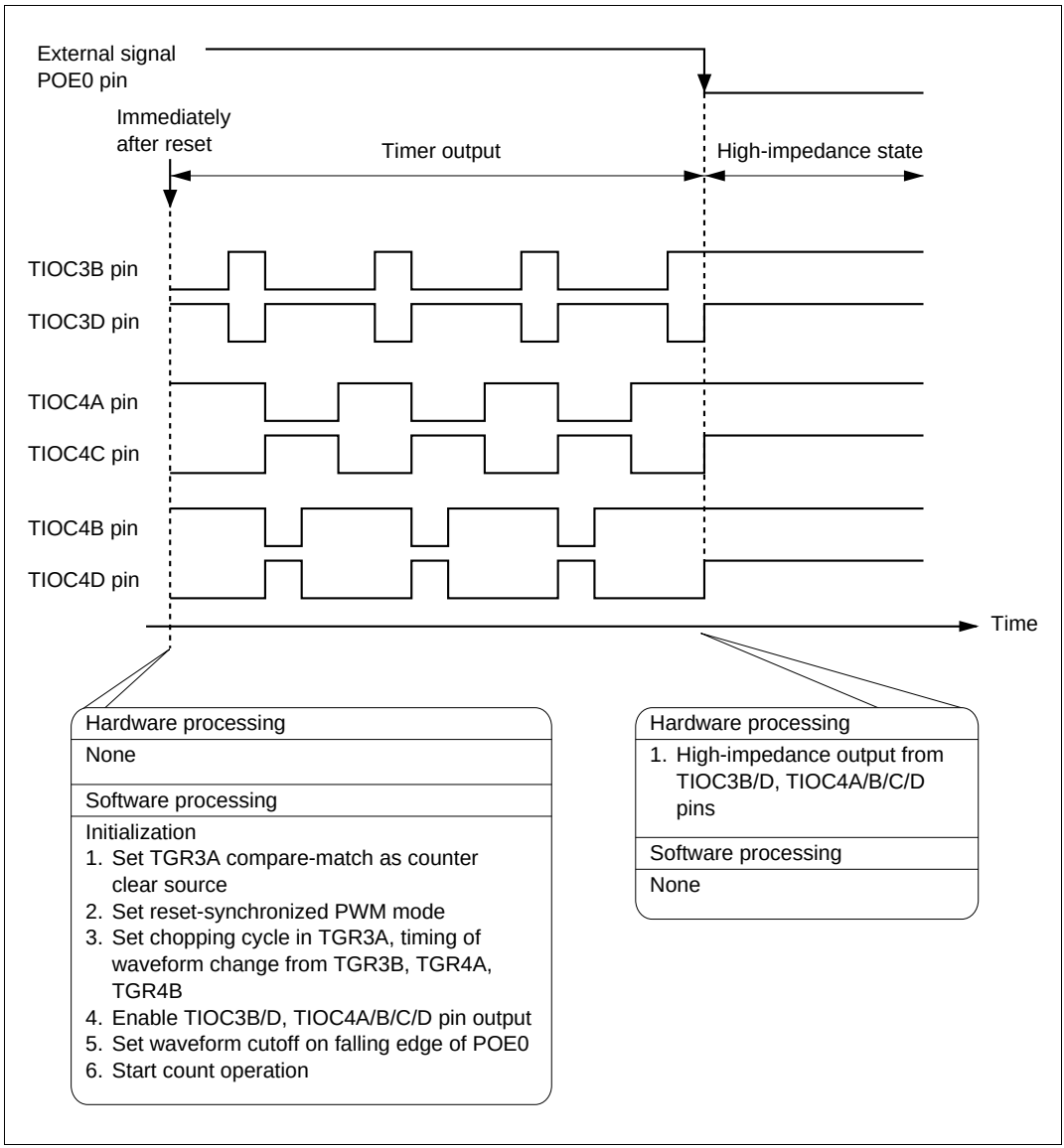


Figure 2-8-3 Principles of Externally Triggered Timer Waveform Cutoff Operation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	down	DC motor control waveform generation

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
cycle	PWM cycle setting	1 word	Main routine	Input
duk1	Setting of timing for change of waveform output from TIOC3B/D			
duk2	Setting of timing for change of waveform output from TIOC4A/C			
duk3	Setting of timing for change of waveform output from TIOC4B/D			

3. Internal Registers Used

Register Name	Function Assignment	Module
TMRSH.TSTR	Timer count operation	Main routine
T3.TCR3	Selection of timer counter clear source and input clock	
T3.TOCR	Enabling of toggle output synchronized with PWM cycle, and positive-phase and opposite-phase output level setting	
T31.TGR3A	PWM cycle setting	
T31.TGR3B	Setting of timing for change of waveform output from TIOC3B, TIOC3D	
T31.TGR4A	Setting of timing for change of waveform output from TIOC4A, TIOC4C	
T31.TGR4B	Setting of timing for change of waveform output from TIOC4B, TIOC4D	
T3.TOER	Sets TIOC3B/D, TIOC4A/B/C/D pins to MTU output	
T3.TMDR3	Reset-synchronized PWM mode setting	
POE.ICSR	Setting to high-impedance output synchronized with falling edge of POE0 pin input signal	
FPCE.PEIOR	Sets TIOC3B/D, TIOC4A/B/C/D pins to output	
FPCE.PECR1	Sets TIOC3B/D, TIOC4A/B/C/D pins to MTU output	

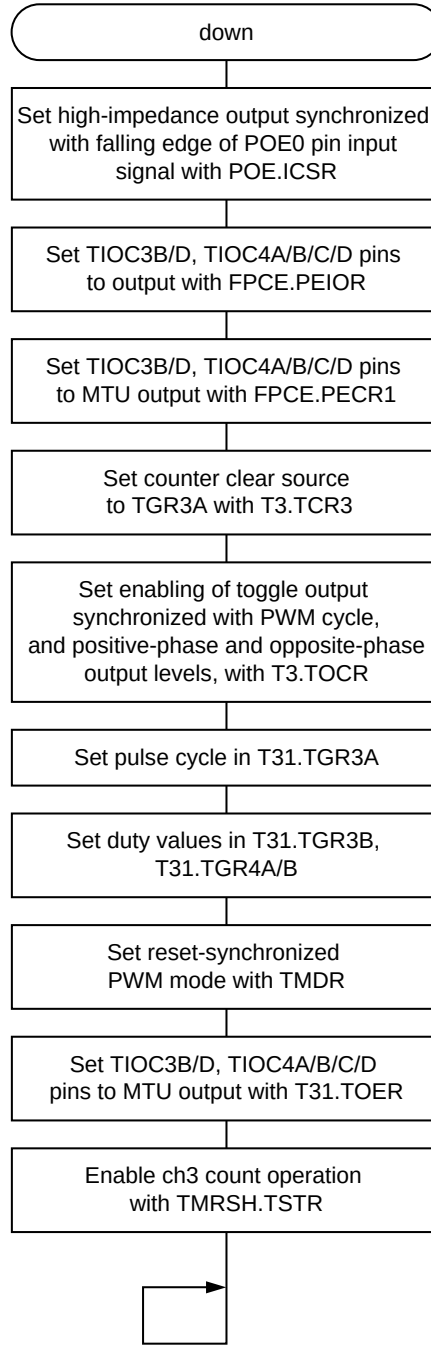
4. RAM Used

This sample task does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



Program List

```
/*---------------------------------------------------------------------*/
/*                            INCLUDE FILE                             */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*---------------------------------------------------------------------*/
/*                             PROTOTYPE                               */
/*---------------------------------------------------------------------*/
void down(void);
/*---------------------------------------------------------------------*/
/*                          RAM ALLOCATION                             */
/*---------------------------------------------------------------------*/
#define cycle   (*(unsigned short *)0xffffe800)
#define duk1    (*(unsigned short *)0xffffe802)
#define duk2    (*(unsigned short *)0xffffe804)
#define duk3    (*(unsigned short *)0xffffe806)
/*---------------------------------------------------------------------*/
/*                            MAIN PROGRAM                             */
/*---------------------------------------------------------------------*/
void down(void)
{
    POE.ICSR = 0x00;            /* stop timer POE0 falling edge */
    POE.OCSR = 0x00;
    PFCE.PEIOR = 0xfa00;        /* set output level */
    PFCE.PECR1 = 0x5544;        /* TIOCnx sellect */
    T0.TCR0 = 0x20;             /* timer clear input capture TGRA0 */
    T0.TIOR0H = 0x88;           /* input capture riseing edge TIOC0A */
    T0.TIOR0L = 0x08;           /* input capture riseing edge TIOC0A */
    T3.TCR3 = 0x20;             /* timer clear input capture TGRA3 */
    T3.TOCR = 0x00;             /* set output level */
    T31.TCNT3 = 0x0000;         /* set timer counter 0x0000 */
    T3.TMDR3 = 0xc8;            /* reset-synchronized pwm mode */
    T31.TGR3A = cycle;          /* cycle set */
    T31.TGR3B = duk1;           /* width set */
    T31.TGR4A = duk2;
    T31.TGR4B = duk3;
    T3.TOER = 0xff;             /* set timer3,4 output */
    TMRSH.TSTR = 0xc0;          /* start timer3,4 */
    while(1);                   /* loop */
}
```

Specifications

1. Waveforms required for DC brushless motor control are output as shown in figure 2-9-1. The waveforms are output by chopping the pin gate signals and reset-synchronized PWM output.

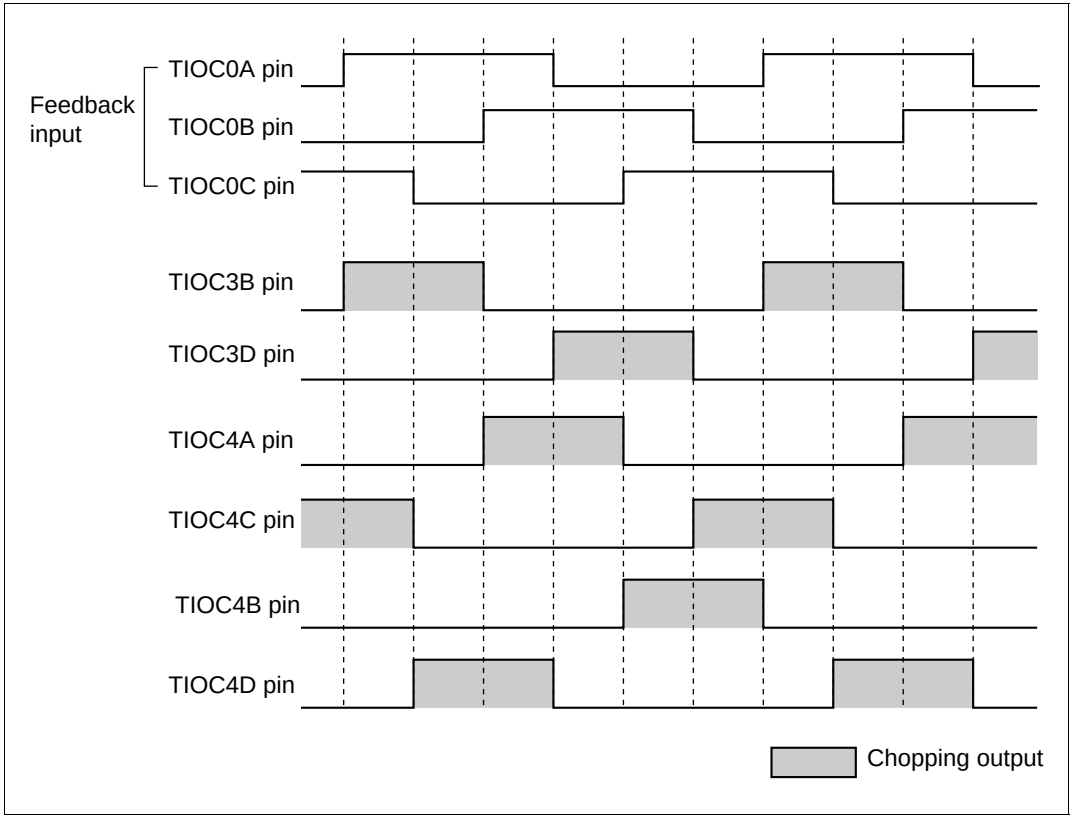


Figure 2-9-1 Example of DC Brushless Motor Control Signal Output

Functions Used

1. In this sample task, 3-phase output is performed of PWM waveforms with a positive-phase/opposite-phase relationship and a common turning point, using MTU ch3 and ch4 in combination. The gate signals generated from the created waveforms and feedback input are chopped and output.

   a. Figure 2-9-2 shows the MTU block diagram for this sample task.

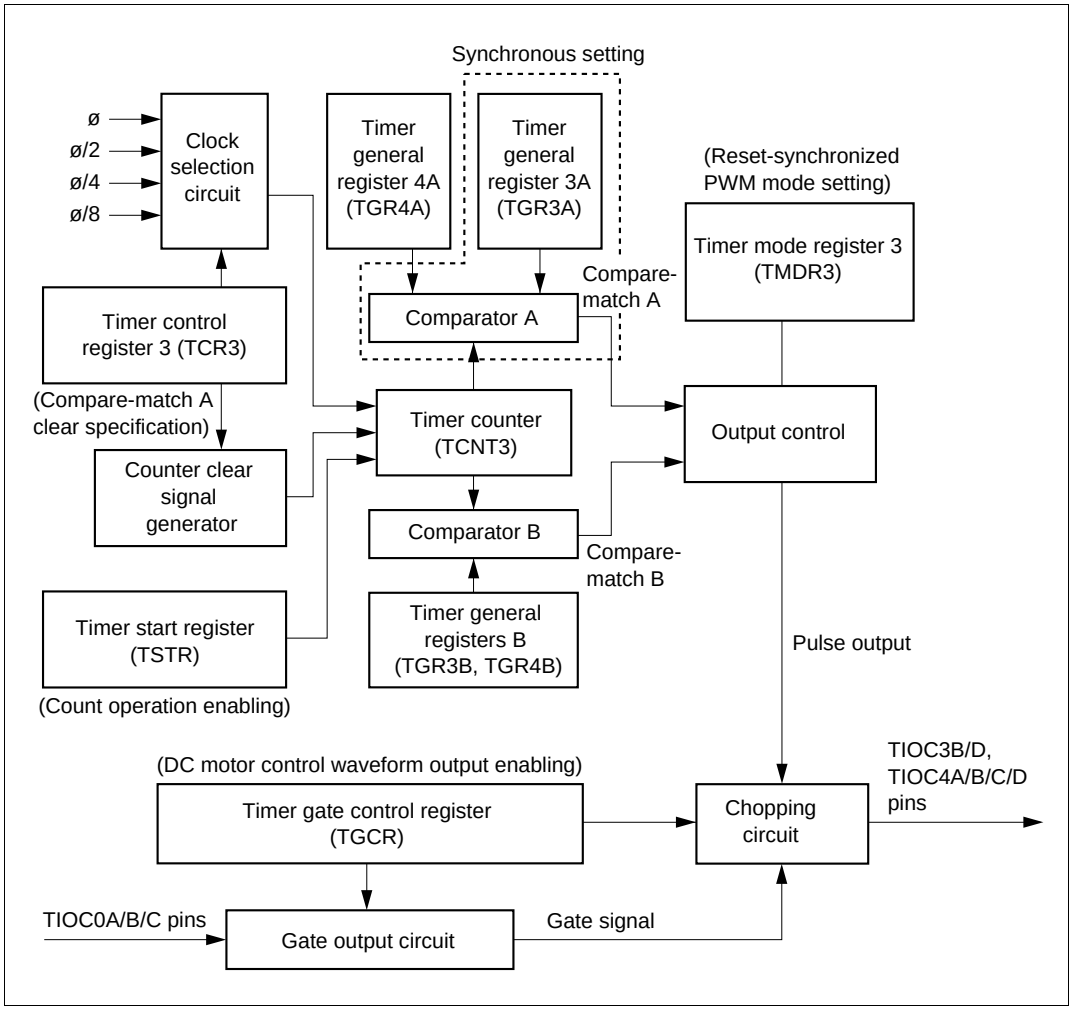


Figure 2-9-2 MTU/ch3, ch4 Block Diagram

2. Table 2-9-1 shows the function assignments for this task. MTU functions are assigned as shown in this table to perform DC motor control waveform output.

Table 2-9-1 Function Assignment

Pin/Register Name	Function Assignment
TIOC3B	Pulse output pins
TIOC3D	
TIOC4A	
TIOC4B	
TIOC4C	
TIOC4D	
TIOC0A	Feedback signal input pins
TIOC0B	
TIOC0C	
TSTR	Enables/disables ch3, ch4 timer counter operation
TCR3	Selects ch3 timer counter clear source and input clock
TMDR3	Sets ch3, ch4 to reset-synchronized PWM mode operation
TGR3A	PWM cycle setting
TGR3B	Output waveform change timing setting
TGR3C	
TGR3D	
TGR4A	
TGR4B	
TOER	Enables/disables TIOC3B/D, TIOC4A/B/C/D pin timer output
TGCR	Enables/disables DC motor control waveform output

Operation

Figure 2-9-3 shows the principles of the operation. DC motor control waveform output is performed automatically by hardware. (For the principles of reset-synchronized PWM operation, see section 2.5, Positive-Phase and Opposite-Phase PWM 3-Phase Output, in this Application Note.)

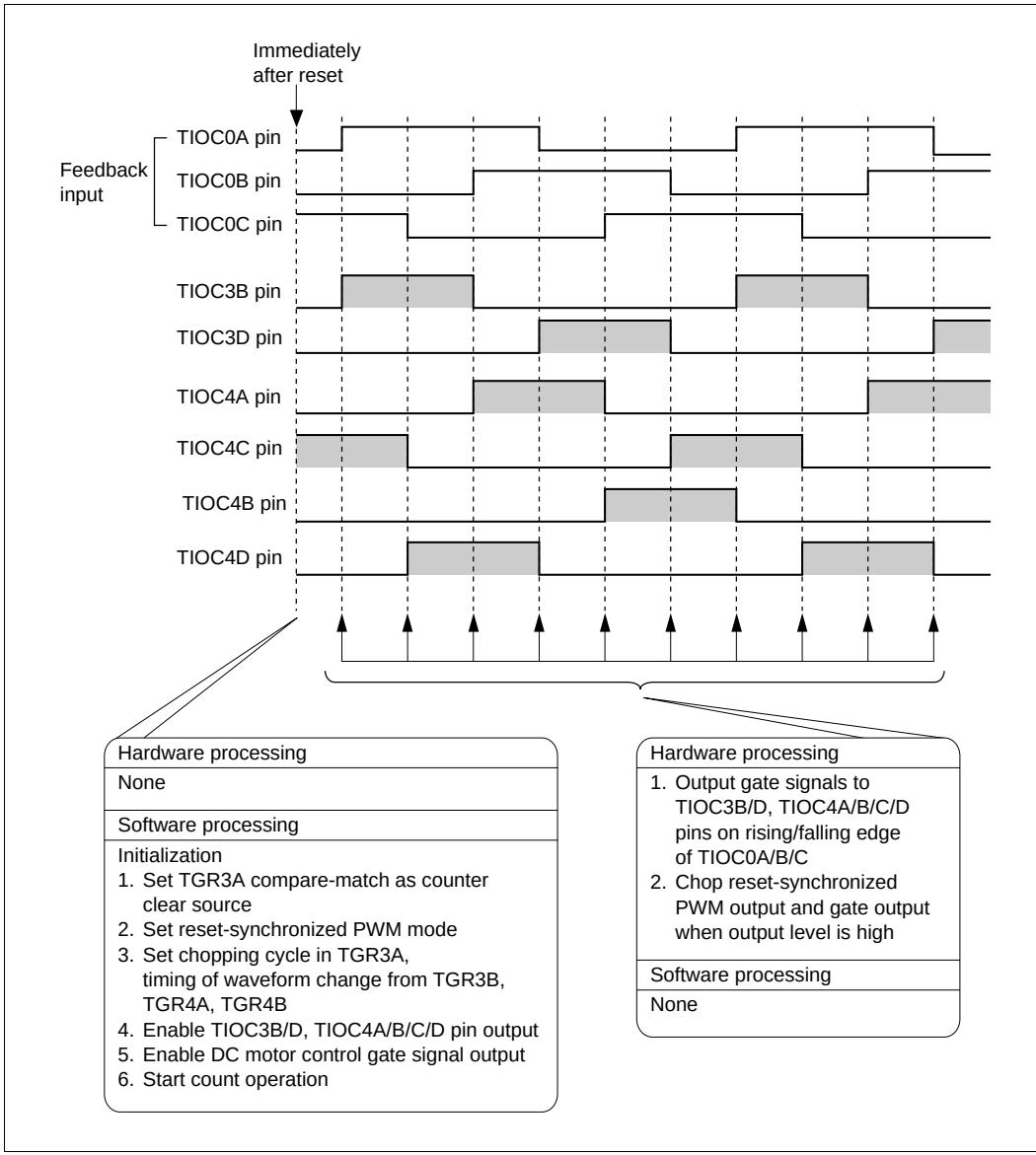


Figure 2-9-3 Principles of DC Motor Control Signal Output Operation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	dc_3out	DC motor control waveform generation

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
cycle	PWM cycle setting	1 word	Main routine	Input
duk1	Setting of timing for change of waveform output from TIOC3B/D			
duk2	Setting of timing for change of waveform output from TIOC4A/C			
duk3	Setting of timing for change of waveform output from TIOC4B/D			

3. Internal Registers Used

Register Name	Function Assignment	Module
FPCE.PEIOR	Sets TIOC3B/D, TIOC4A/B/C/D pins to output	Main routine
FPCE.PECR1	Sets TIOC3B/D, TIOC4A/B/C/D pins to MTU output	
TMRSH.TSTR	Timer count operation/halt setting	
T3.TCR3	Selection of timer counter clear source and input clock	
T3.TOCR	Enabling of toggle output synchronized with PWM cycle, and positive-phase and opposite-phase output level setting	
T31.TGR3A	PWM cycle setting	
T31.TGR3B	Setting of timing for change of waveform output from TIOC3B, TIOC3D	
T31.TGR4A	Setting of timing for change of waveform output from TIOC4A, TIOC4C	
T31.TGR4B	Setting of timing for change of waveform output from TIOC4B, TIOC4D	
T3.TOER	Sets TIOC3B/D, TIOC4A/B/C/D pins to MTU output	
T3.TMDR3	Reset-synchronized PWM mode setting	
T3.TGCR	Enables DC motor control waveform output	

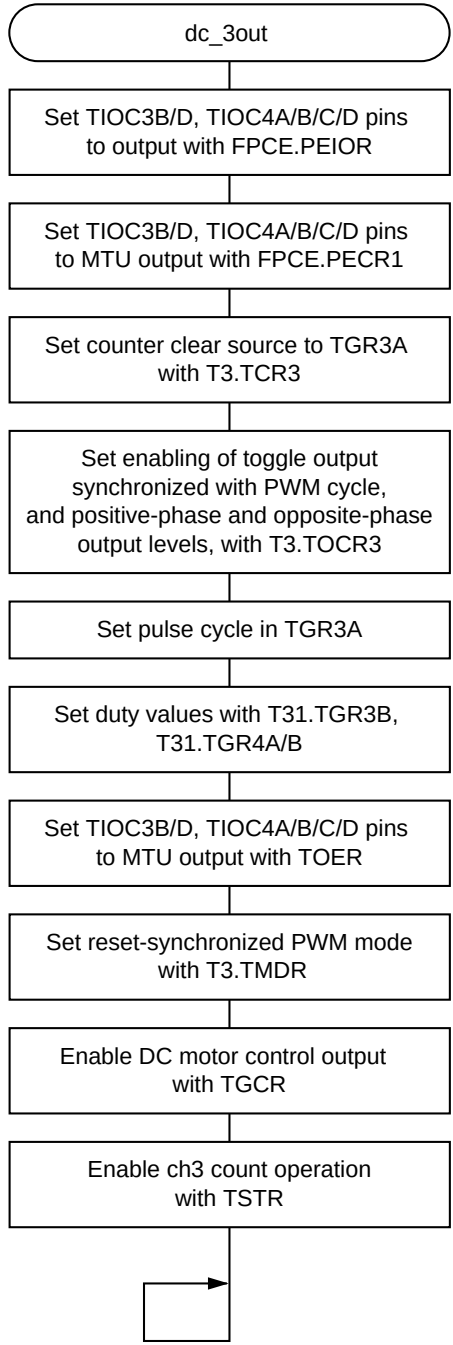
4. RAM Used

This sample task does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



Program List

```
/*---------------------------------------------------------------------*/
/*                            INCLUDE FILE                             */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*---------------------------------------------------------------------*/
/*                             PROTOTYPE                               */
/*---------------------------------------------------------------------*/
void dc_3(void);
/*---------------------------------------------------------------------*/
/*                           RAM ALLOCATION                            */
/*---------------------------------------------------------------------*/
#define cycle   (*(unsigned short *)0xffffe800)
#define duk1    (*(unsigned short *)0xffffe802)
#define duk2    (*(unsigned short *)0xffffe804)
#define duk3    (*(unsigned short *)0xffffe806)
/*---------------------------------------------------------------------*/
/*                            MAIN PROGRAM                             */
/*---------------------------------------------------------------------*/
void dc_3(void)
{
    PFCE.PEIOR = 0xfa00;     /* set output level */
    PFCE.PECR1 = 0x5544;     /* TIOCnx sellect */
    PFCE.PECR2 = 0x0055

    T0.TCR0 = 0x20;          /* timer clear input capture TGRA0 */
    T3.TCR3 = 0x20;          /* timer clear input capture TGRA3 */
    T3.TOCR = 0x00;          /* set output level */
    T31.TCNT3 = 0x0000;      /* set timer counter 0x0000 */
    T3.TMDR3 = 0xc8;         /* reset-synchronized pwm mode */
    T31.TGR3A = cycle;       /* cycle set */
    T31.TGR3B = duk1;        /* width set */
    T31.TGR4A = duk2;
    T31.TGR4B = duk3;
    T3.TOER = 0xff;          /* set timer3,4 output */
    T3.TGCR = 0x70;          /* set DC motor output */
    TMRSH.TSTR = 0xc1;       /* start timer0,3,4 */
    while(1);                /* loop */
}
```

2.10 Activation of A/D Conversion by MTU and Storage of Conversion Results

DTC (Block Transfer), A/D Converter

Specifications

1. Voltages are input on 8 channels as shown in figure 2-10-1, and the A/D conversion results are stored in RAM by the DTC.
2. Group mode or single mode can be set for A/D conversion, with 2 channels sampled simultaneously.
3. A/D conversion is started by an MTU TGR0A compare-match.
4. DTC block transfer is performed by means of A/D end interrupts.

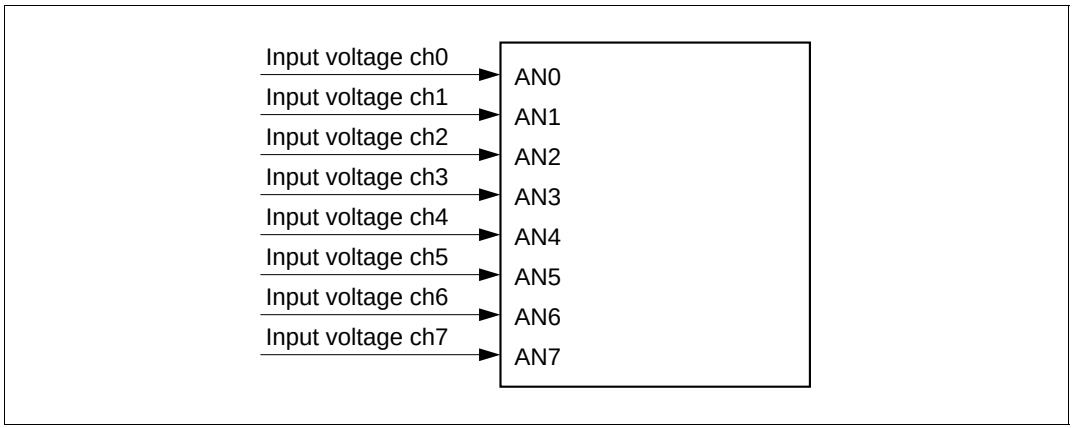


Figure 2-10-1 Block Diagram of Voltage Measurement by SH7040

Functions Used

1. In this sample task, A/D conversion is started by an MTU compare-match, and the conversion results are stored in RAM by the DTC.

   a. Figure 2-10-2 shows the ch0 block diagram. Initiation of A/D conversion is performed using the following functions:

      - A function for starting A/D conversion upon MTU compare-match without software intervention
      - A function for automatically outputting pulses by hardware without software intervention (output compare)

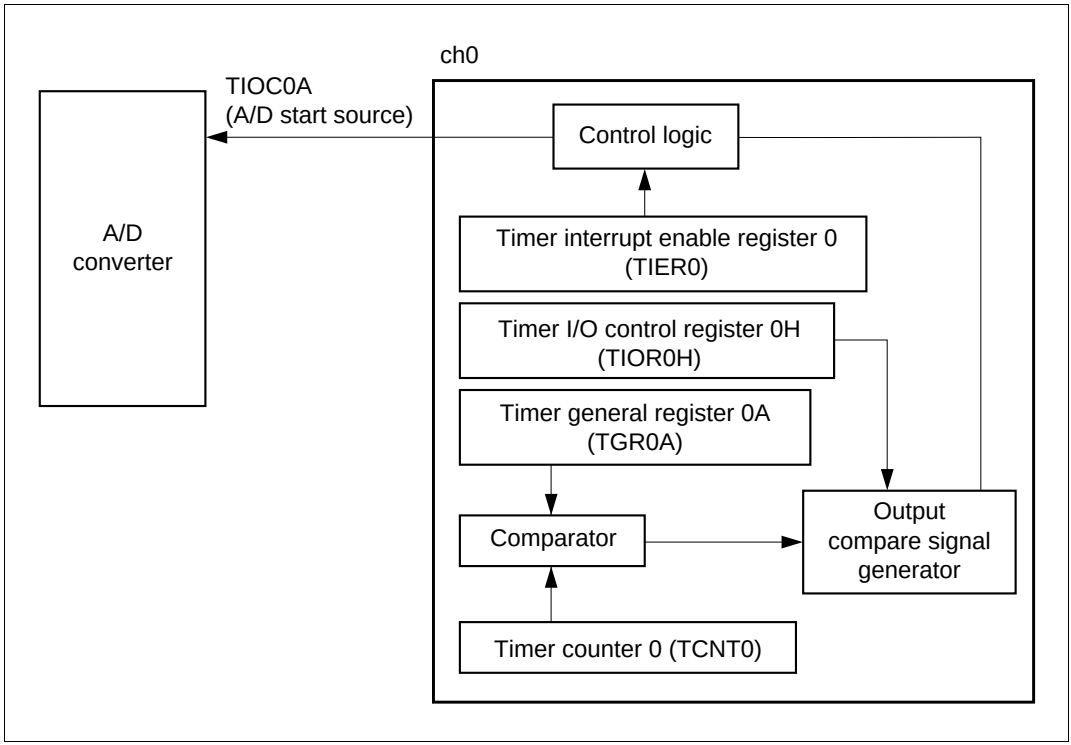


Figure 2-10-2 SH7040 ch0 Block Diagram

b. Figure 2-10-3 shows the A/D converter block diagram. The A/D converter performs analog-to-digital conversion using the following functions. The DTC is activated at the end of A/D conversion.

- A function that performs A/D conversion on a number of channels (chA to chH) (group/single mode)
- A function that performs continuous conversion by sampling the input voltages on two channels simultaneously (simultaneous sampling)
- A function that activates the DTC at the end f A/D conversion

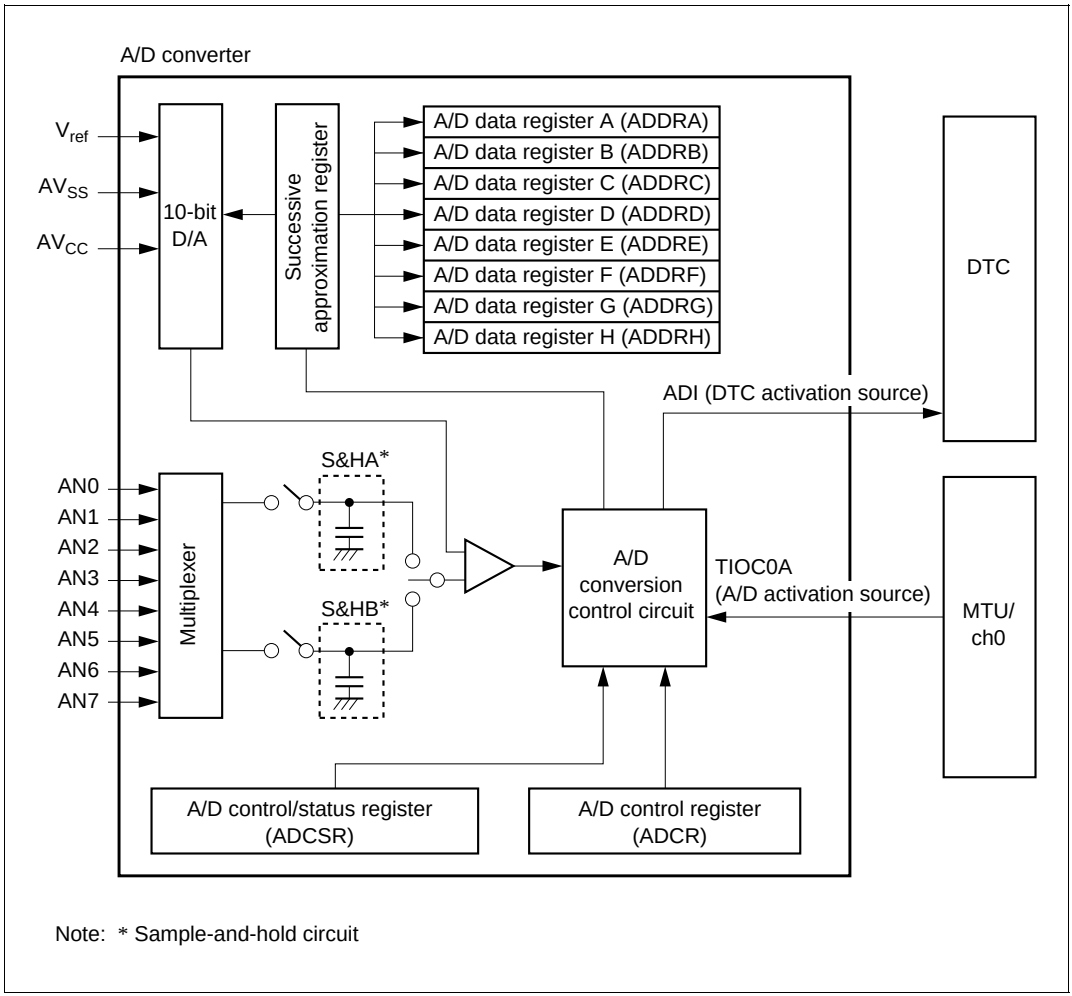


Figure 2-10-3 Block Diagram of Voltage Measurement by SH7040

c. Figure 2-10-4 shows the DTC block diagram. The DTC performs block transfer using the following function:

- A function that transfers one block of data in response to one activation source (block transfer function)

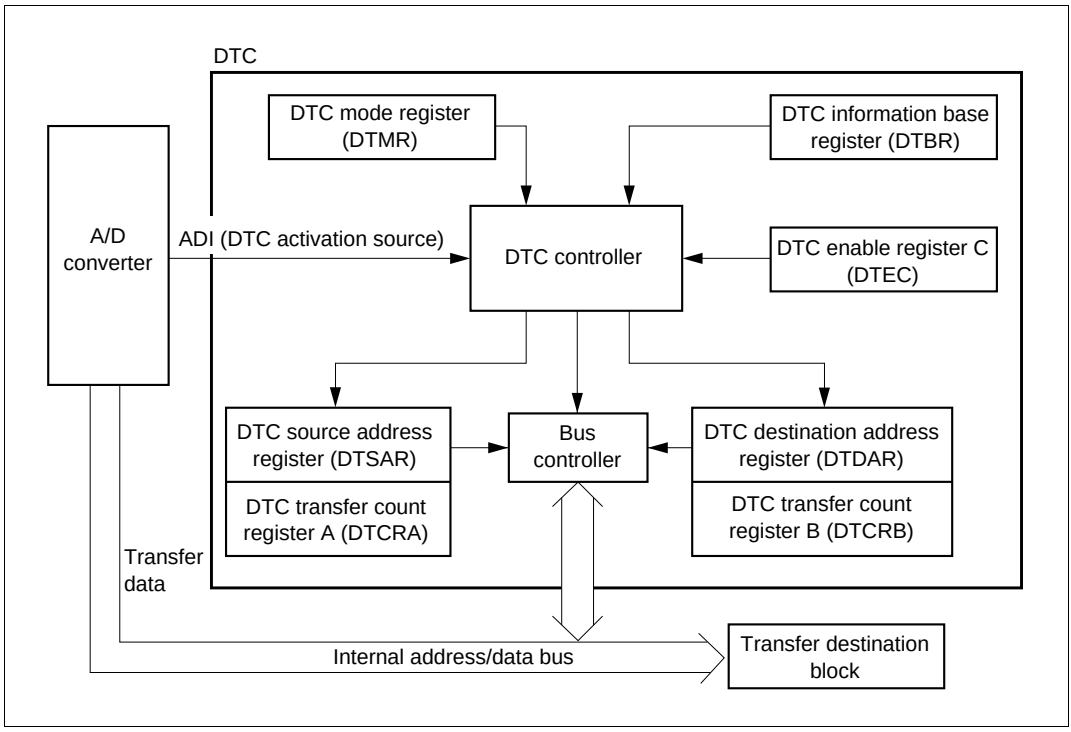


Figure 2-10-4 SH7040 DTC Block Diagram

2. Table 2-10-1 shows the function assignments for this task.

Table 2-10-1Function Assignment

Pin/Register Name	Function Assignment
A0 to A7	Analog measurement pins
TCR0	Counter clear source selection
TIOR0H	Input capture signal input edge selection
TIER0	Enables A/D conversion start request generation
TGR0A	Sampling cycle setting
ADCR	A/D conversion mode and measurement pin setting
ADCSR	Conversion time and activation source selection
ADDRA to ADDRH	Store A/D conversion results
DTMR	Sets DTC to block transfer mode
DTCRA	Transfer number specification
DTCRB	Block length specification
DTSAR	Transfer source address setting
DTDAR	Transfer destination address setting
DTBR	DTC vector upper 16 bit setting
DTEC	Enables DTC activation at end of A/D conversion

Operation

Figure 2-10-5 shows the principles of the operation. As shown in the figure, A/D conversion is started by a TGR0A compare-match, input voltages on AN0 to AN7 are sampled 2 channels at a time, and conversion is performed sequentially on AN0 to AN7. When conversion is completed for all the specified channels, the DTC is activated and the A/D conversion results are transferred to RAM.

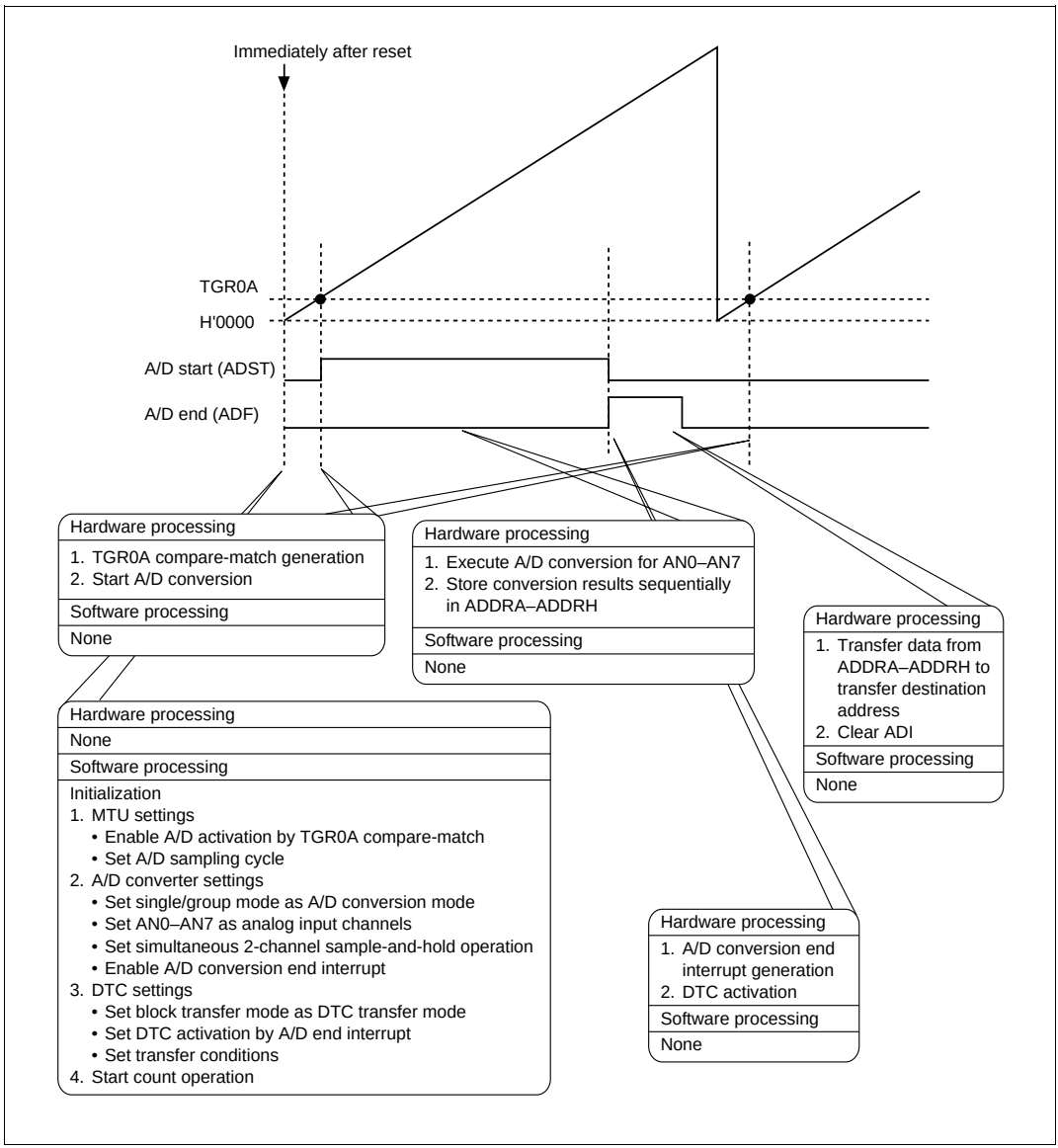


Figure 2-10-5 Principles of Pulse Width Measurement Operation

Figure 2-10-6 shows the principles of the DTC activation operation. When the DTC is operated, the following settings should be made before activation source generation.

- Register information settings. (Sample register information settings for block transfer mode are shown in table 2-11-2.)
- Set the lower 16 bits of the start address in which the register information is set in the DTC vector table.
- Set the upper 16 bits of the start address in which the register information is set in the DTC information base register.

As shown in figure 2-10-6, when the DTC is activated, the lower 16 bits of the register information start address are read from the DTC vector table. The register information start address is generated from the value read and the DTC information base register (holding the upper 16 bits of the register information start address). Register information is read and transferred sequentially starting from the register information start address.

Table 2-11-2 Register Information

Address	Register Name	Data Length
RF	DTC mode register (DTMR)	Word
RF+2	DTC transfer count register A (DTCRA)	Word
RF+6	DTC transfer count register B (DTCRB)	Word
RF+8	DTC source address register (DTSAR)	Longword
RF+12	DTC destination address register (DTDAR)	Longword

RF: Register information start address

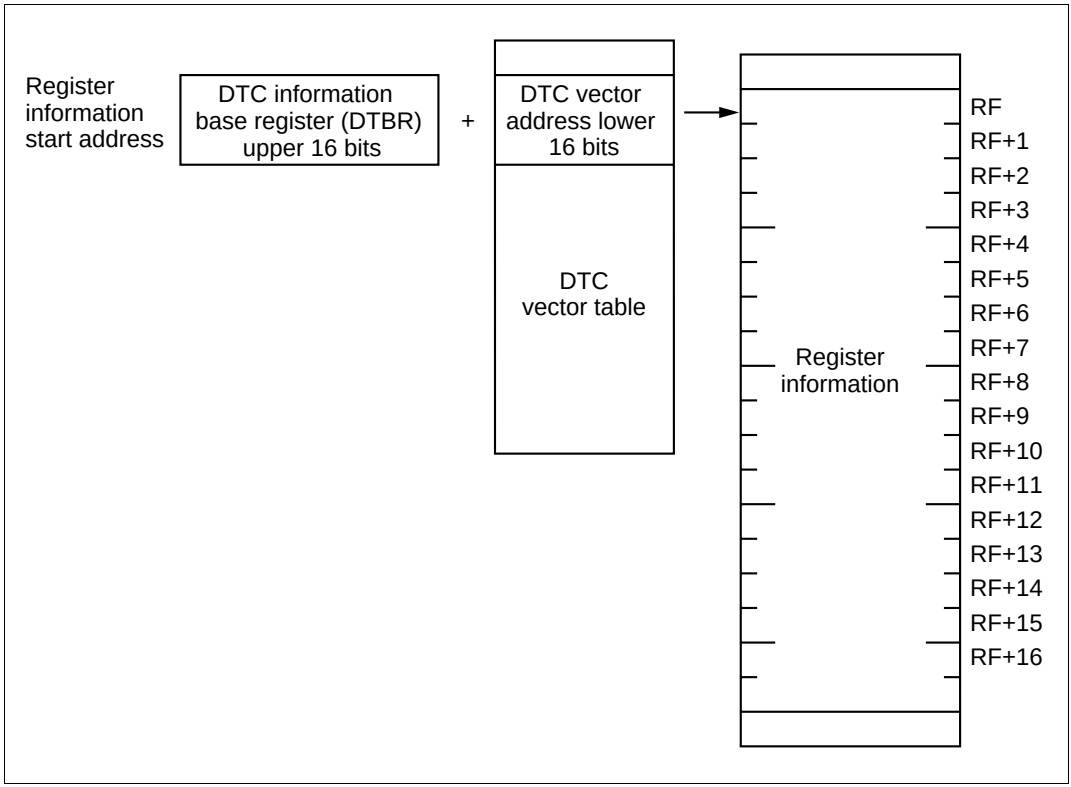


Figure 2-10-6 Principles of the DTC Activated Operation

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	adscan	Setting of A/D converter activation by MTU and DTC activation at end of A/D conversion

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
transsadr	DTC transfer destination address setting For 8 channel A/D conversion results, 8 data items are stored in longword units starting with transaddr Each 10-bit conversion result is set as follows:	Longword	Main routine	Input

Upper byte							AD9	AD8
Lower byte	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

3. Internal Registers Used

Register Name	Function Assignment	Module
TMRSH0.TSTR	Timer count operation/disable setting	Main routine
T0.TCR0	TCNT counter clock selection and setting of output compare A as counter clear source	
T0.TIOR0H	Sets TGR0A to output compare	
T0.TIER0	Enables A/D conversion start request generation	
T0.TGRA0	Sets A/D conversion sampling cycle	
A_D.ADCR	Setting of A/D conversion mode (single mode), simultaneous sampling operation, and activation source as MTU conversion start trigger	
A_D.ADCSR	A/D conversion channel (group mode), conversion channel (AN0–AN7), and conversion time setting Enables A/D conversion end interrupt	
A_D.ADDRA to A_D.ADDRH	A/D conversion result storage	—
DTC.DTMR	Sets DTC to block transfer mode	Main routine
DTC.DTCRA	Sets number of transfers to 1	
DTC.DTCRB	Specifies block length of 8	
DTC.DTSAR	Sets ADDRA as transfer destination address	
DTC.DTDAR	Sets RAM as transfer destination address	
DTC.DTBR	Setting of DTC vector upper 16 bits	
DTC.DTEC	Enables DTC activation at end of A/D conversion	
PFCE.PECR2	Enables pulse input from TIOC0A	
INTC.IPRG	Sets A/D conversion end interrupt level to 15	

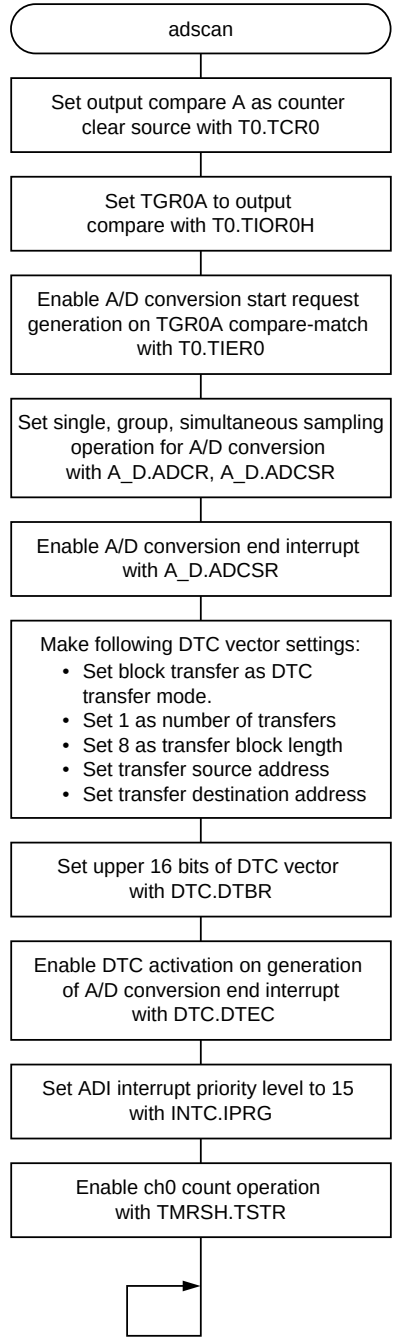
4. RAM Used

This sample task does not use any RAM apart from the arguments.

Note: The SH7040 header file name is used as the register label.

Flowchart

1. Main routine



Program List

```
/*---------------------------------------------------------------------*/
/*                            INCLUDE FILE                             */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"SH7040.H"
/*---------------------------------------------------------------------*/
/*                             PROTOTYPE                               */
/*---------------------------------------------------------------------*/
void a_d(void);
/*---------------------------------------------------------------------*/
/*                           RAM ALLOCATION                            */
/*---------------------------------------------------------------------*/
#define DTMR    (*(unsigned short *)0xffffe800)
#define DTCRA   (*(unsigned short *)0xffffe802)
#define DTCRB   (*(unsigned short *)0xffffe806)
#define DTSAR   (*(unsigned long *)0xffffe808)
#define DTDAR   (*(unsigned long *)0xffffe80c)
#define dtmr1   (*(unsigned short *)0xffffe810)
#define dtcra1  (*(unsigned short *)0xffffe812)
#define dtcrb1  (*(unsigned short *)0xffffe816)
#define dtsar1  (*(unsigned long *)0xffffe818)
#define dtdar1  (*(unsigned long *)0xffffe81c)
/*---------------------------------------------------------------------*/
/*                             MAIN PROGRAM                            */
/*---------------------------------------------------------------------*/
void a_d(void)
{
    T0.TCR0 = 0x20;          /* timer clear output compare TGR0A */
    T0.TIOR0H = 0x00;
    T0.TIER0 = 0x80;         /* interrupt TGI00A */
    T0.TGR0A = 0x1000;       /* set get A/D cycle */
    A_D.ADCR = 0x14;         /* set sampling */
    A_D.ADCSR = 0x4f;        /* set mode single/group */
    DTMR = dtmr1;            /* set transmission mode DTC */
    DTCRA = dtcra1;          /* set transmission frequency */
    DTCRB = dtcrb1;          /* set transmission block frequency */
    DTSAR = dtsar1;          /* set transmission address */
    DTDAR = dtdar1;          /* set transmission address */
    DTC.DTBR = 0xffff;       /* set DTC data base register */
    DTC.DTEC = 0x40;         /* set a/d end start DTC */
    INTC.IPRG = 0x0f00;      /* set interrupt level=15 */
    set_imask(0x0);          /* set imask level=0 */
    TMRSH.TSTR = 0x01;       /* TCNTO start */
    while(1);                /* loop */
}
```

Note: Set the lower 16 bits of the register information start address in DTC vector table address 0x422 (vector when DTC activation condition is A/D conversion end).

Specifications

1. Using the SH7040's SCI in asynchronous mode, a RAM reference address (4-byte data) transmitted from the console is received, and its contents are fetched from RAM and transmitted to the console via the SCI, as shown in figure 2-11-1.

2. The transmission protocol is 9600 bps, 8-bit data, one stop bit, non-parity.

3. DMAC direct address mode is used for data transfer from RDR to RAM, and the RDR receive data is stored in RAM, as shown in figure 2-12-2.

4. DMAC indirect address mode is used for data transfer from RAM to TDR, as shown in figure 2-11-3. The operation is performed as follows:

   a. The data held in RAM is stored in a temporary buffer in the DMAC, and data is fetched from RAM using the buffer data as the address.

   b. The fetched data is transferred sequentially to TDR in byte units.

5. The DMAC transfer conditions are shown in tables 2-11-1 and 2-11-2.

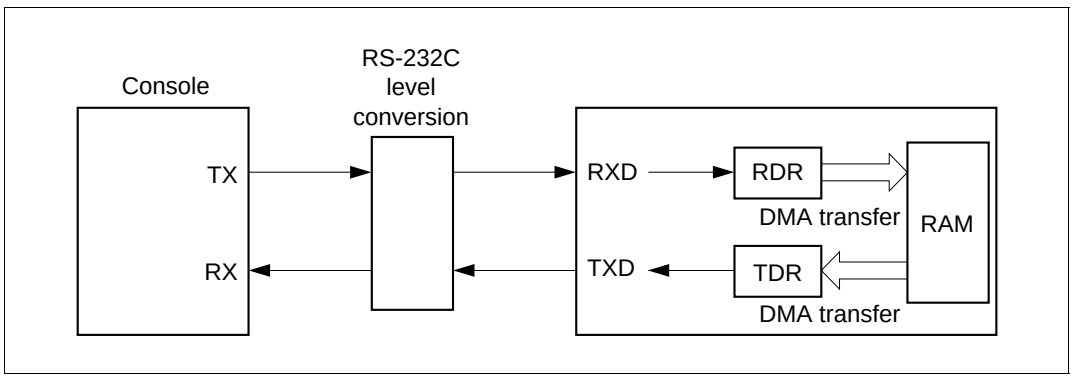


Figure 2-11-1 Block Diagram of SCI Transfer of RAM Data by SH7040

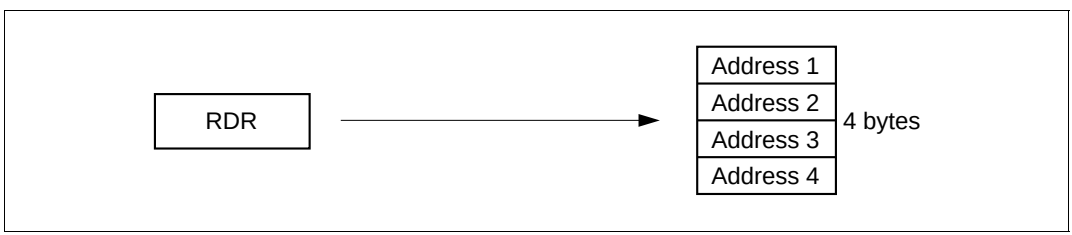


Figure 2-11-2 Data Transfer Using DMAC (Transfer Source Direct Address)

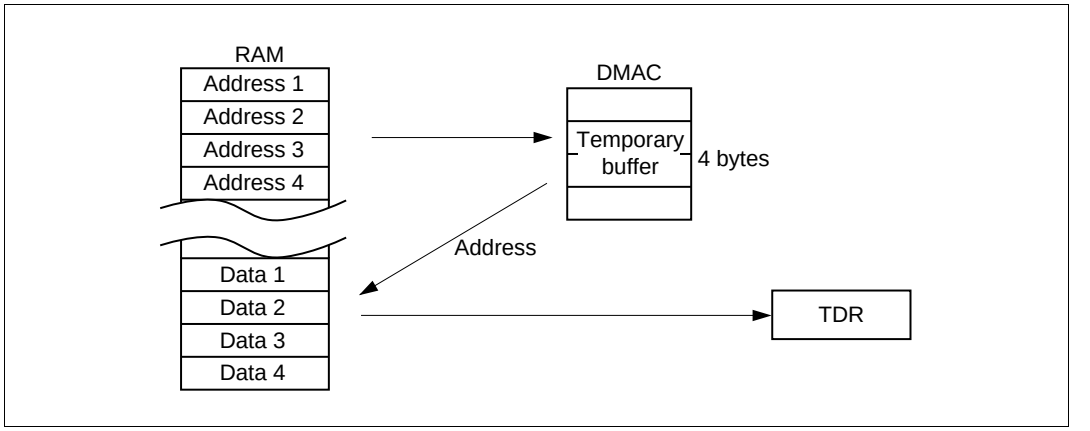


Figure 2-11-3 Data Transfer Using DMAC (Transfer Source Indirect Address)

Table 2-11-1 DMAC Transfer Conditions in SCI Reception (RDR → RAM)

Condition	Description
DMAC channel	Channel 0
Transfer source	On-chip SCI channel 0
Transfer destination	On-chip RAM
Number of transfers	4
Transfer source address	Fixed
Transfer destination address	Incremented
Transfer request source	SCI channel 0
Bus mode	Cycle steal
Transfer unit	Byte

Table 2-11-2 DMAC Transfer Conditions in SCI Transmission (RAM → TDR)

Condition	Description
DMAC channel	Channel 3
Transfer source	On-chip RAM
Transfer destination	On-chip SCI channel 0
Number of transfers	4
Transfer source address	Incremented
Transfer destination address	Fixed
Transfer request source	SCI channel 0
Bus mode	Cycle steal
Transfer unit	Byte

Functions Used

1. This sample task performs RAM monitoring using the SCI and DMAC.

   a. Figure 2-11-4 shows the SCI transmission block diagram. This task performs data transmission to the console using the following SCI functions:

      - A function that carries out data communication asynchronously, with synchronization performed character by character (asynchronous mode)
      - A function that generates an interrupt on completion of transmission (TEI interrupt)

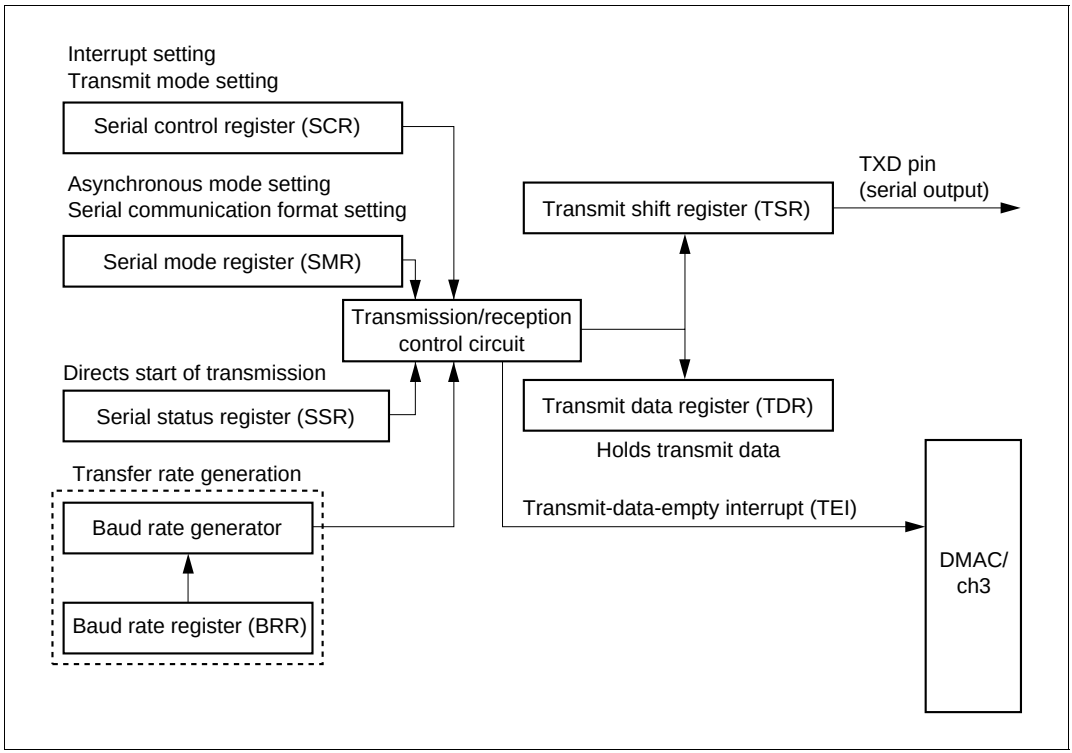


Figure 2-11-4 SCI Transmission Block Diagram

b. Figure 2-11-5 shows the SCI reception block diagram. This task performs data reception from the console using the following SCI functions, as shown in the figure:

- A function that carries out data communication asynchronously, with synchronization performed character by character (asynchronous mode)
- A function that generates an interrupt on completion of reception (RXI interrupt)

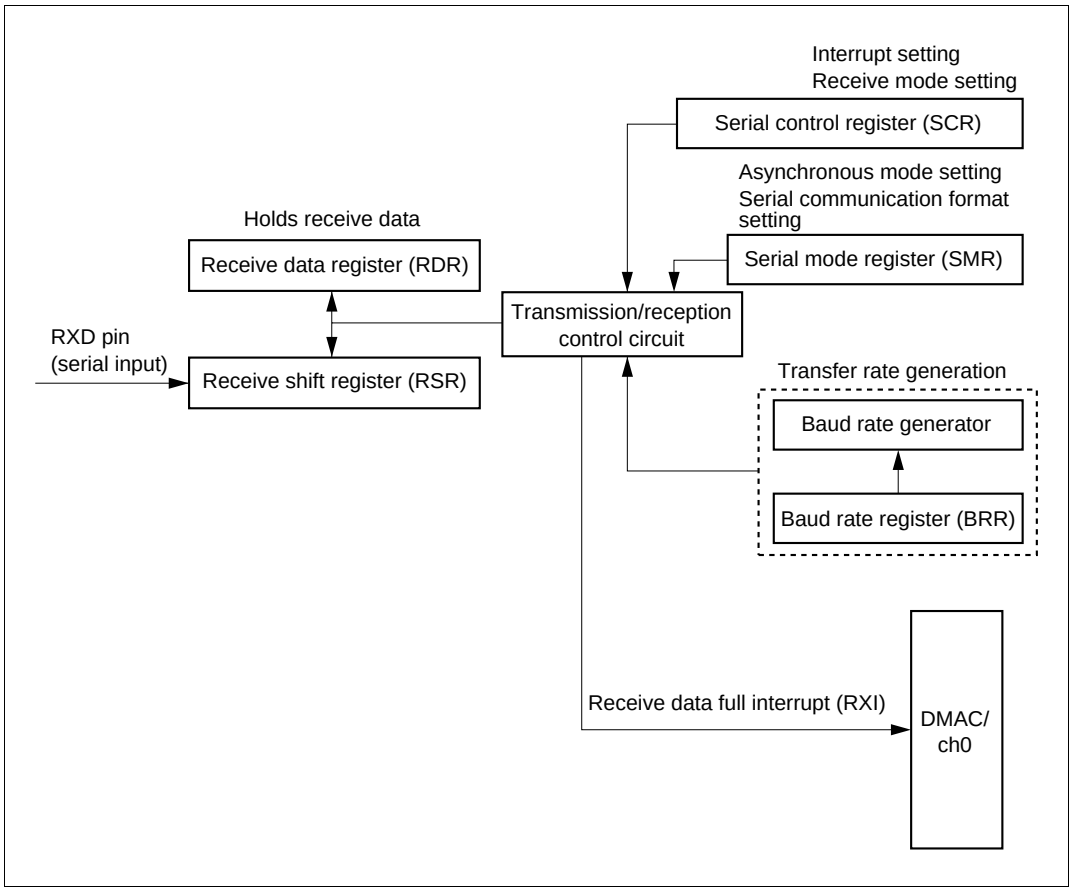


Figure 2-11-5 SCI Reception Block Diagram

c. Figure 2-11-6 shows the DMAC/ch0 block diagram for this sample task. This task performs block transfer using the following DMAC/ch0 functions:

- A function that directly transfers data for both the transfer source and transfer destination when the DMAC is activated (direct address transfer mode)
- A function that activates the DMAC via the SCI

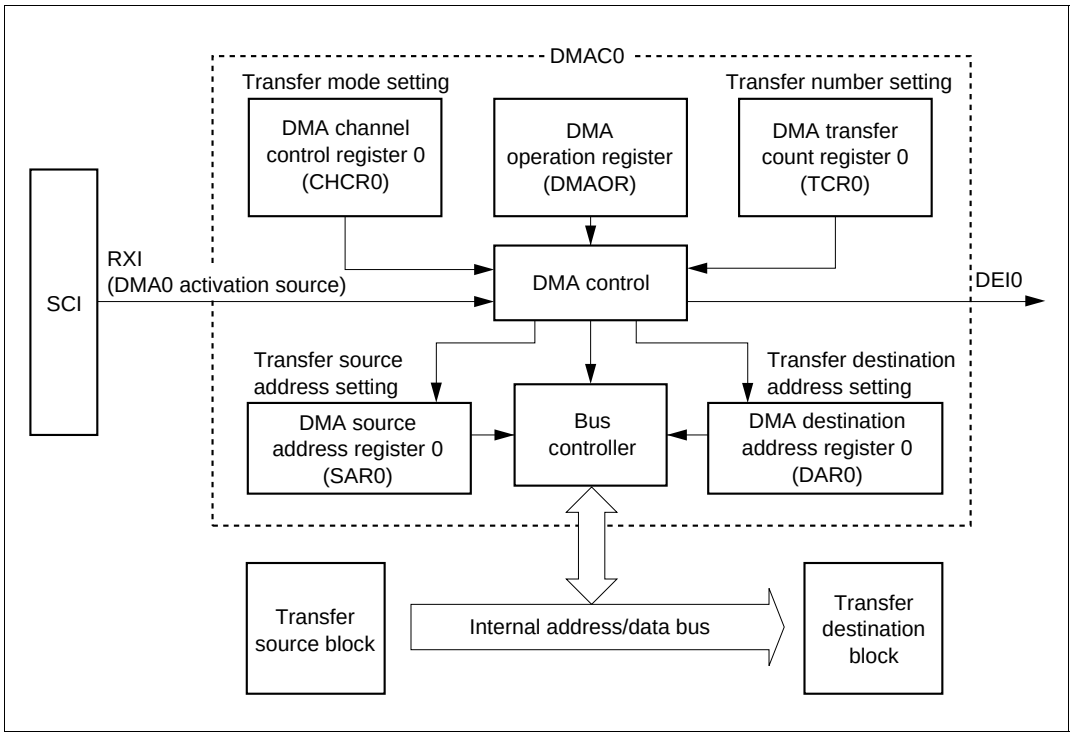


Figure 2-11-6 DMA/ch0 Block Diagram

d. Figure 2-11-7 shows the DMAC/ch3 block diagram for this sample task. This task performs block transfer using the following DMAC/ch3 functions:

- A function that transfers data stored in the transfer source register when the DMAC is activated (indirect address transfer mode)
- A function that activates the DMAC via the SCI

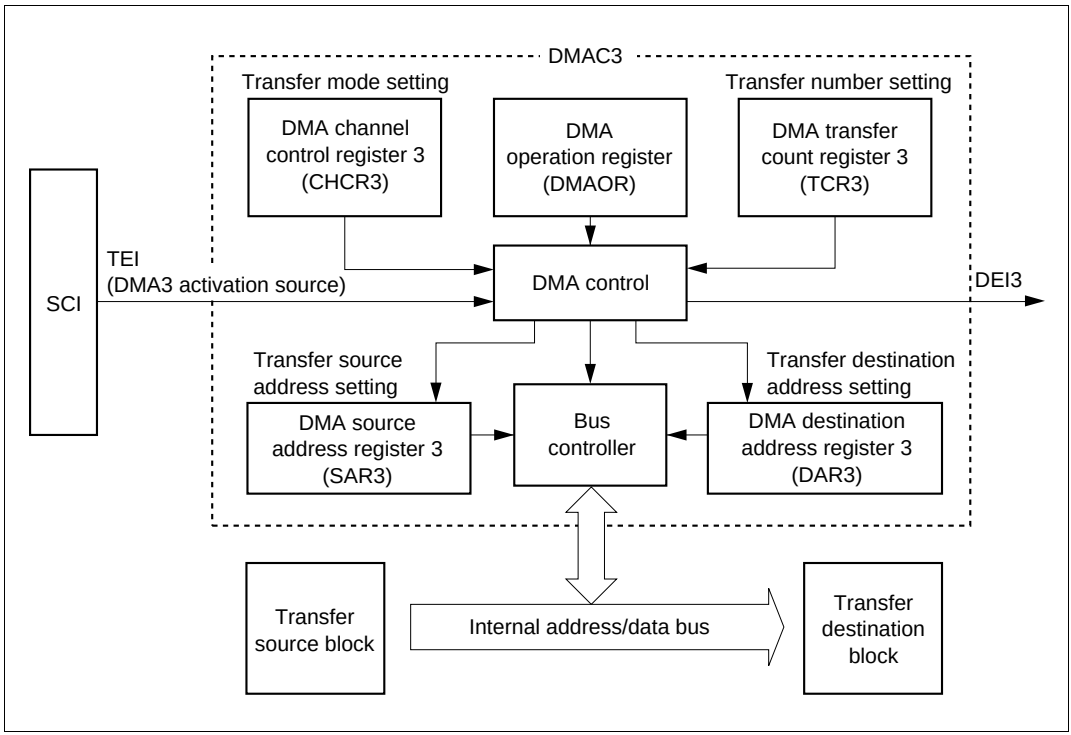


Figure 2-11-7 DMA/ch3 Block Diagram

Table 2-11-1 shows the function assignments for this sample task. DMAC and SCI functions are assigned to perform data transfer transmission/reception via the SCI.

Table 2-11-1 Function Assignment

Pin/Register Name	Function Assignment
SAR0	Transfer source address setting
SAR3	
DAR0	Transfer destination address setting
DAR3	
TCR0	Transfer number setting
TCR3	
CHCR0	Setting of DMAC operating mode, transfer method, etc.
CHCR3	
DMAOR	Executing DMAC channel priority level setting
RXD	Data receive pin
TXD	Data transmit pin
SMR	SCI transmission format setting
SCR	SCI interrupt enabling/disabling setting
SSR	Interrupt status setting
RDR	Holds receive data from console
TDR	Holds transmit data to be sent to console
BRR	Transfer rate setting

Operation

Figure 2-11-9 shows the principles of the operation. As shown in this figure, data transmission and reception is performed serially by means of SH7040 hardware and software processing.

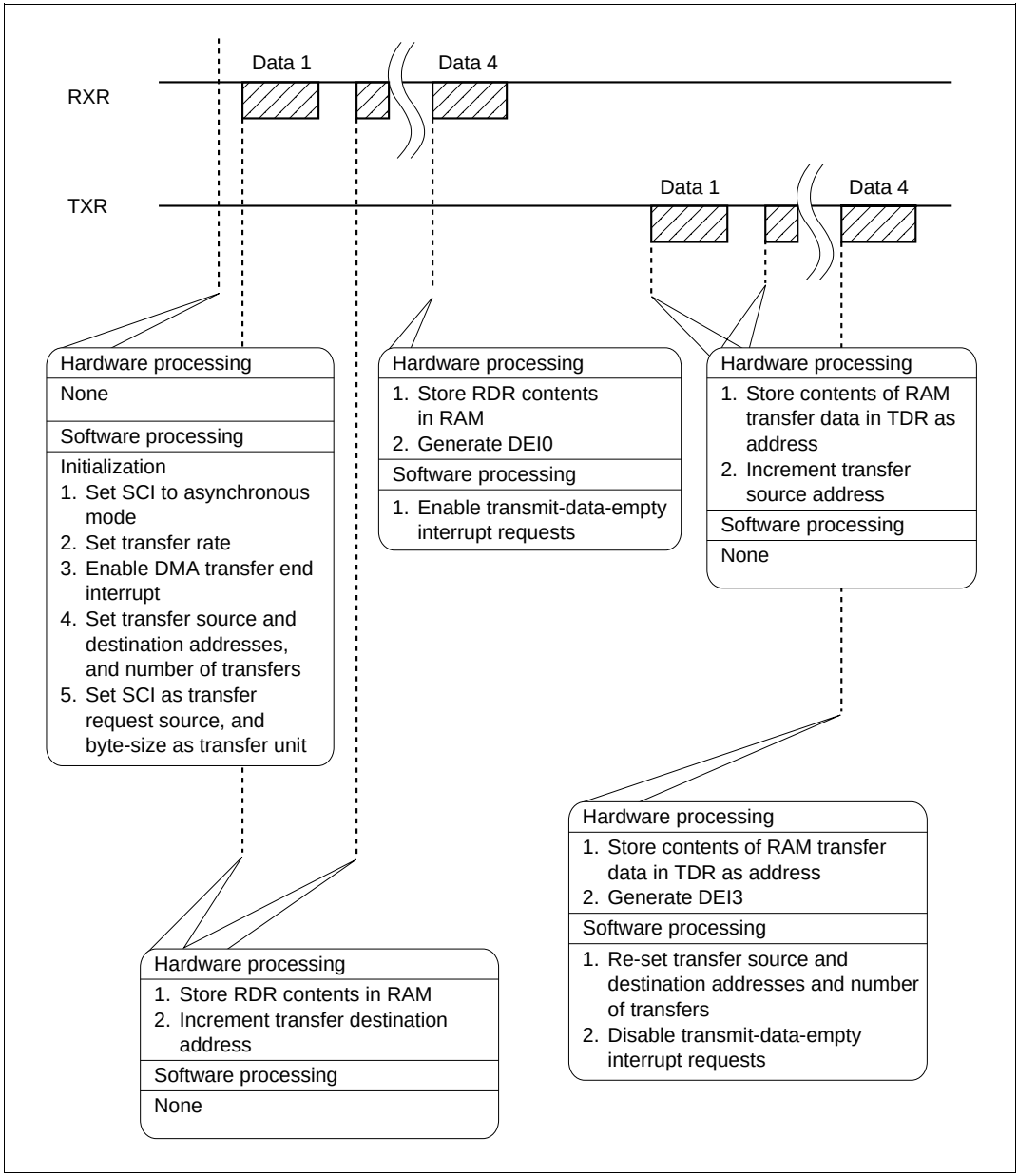


Figure 2-11-8 Principles of Data Transfer Operation Using SCI

Software

1. Modules

Module Name	Label	Function Assignment
Main routine	rammon	Performs SCI and DMAC initialization
Receive data transfer	dma_rdr	Initiated by DEI0; enables transmit-data-empty interrupt requests
Transmit data transfer	dma_tdr	Initiated by DEI3; re-sets transfer source and destination addresses and number of transfers Disables transmit-data-empty interrupt requests

2. Arguments

Label/ Register Name	Function Assignment	Data Length	Module	Input/ Output
data0	Stores transfer address	Longword	Main routine	Input

3. Internal Registers Used

Register Name	Function Assignment	Module
DM0.SAR0	RDR address setting	Main routine
DM0.DAR0	Transfer destination RAM start address setting	Main routine
DM0.TCR0	Set to H'4 (4 transfers)	
DM0.CHCR0	Setting of DMAC operating mode, transfer method, etc	Main routine
DM3.SAR3	Transfer source RAM start address setting	Receive data transfer
DM3.DAR3	TDR address setting	Main routine
DM3.TCR3	Set to H'4 (4 transfers	
DM3.CHCR3	Setting of DMAC operating mode, transfer method, etc.	Main routine
DMAC.DMAOR	Executing DMAC channel priority level setting	Transmit data transfer
PFCA.PADRL	Specifies SCI input/output	
PFCA.PACRL2	Sets pin multiplexing to SCI0 use	
IINTC.IPRC	Sets DMAC0, DMAC3 interrupt priority levels to 14, 15.	
SCI0.SMR0	Sets SCI to asynchronous mode	
SCI0.SCR0	Enables transmit and receive interrupts, and transmit and receive operations	
SCI0.BRR0	Transfer rate setting	

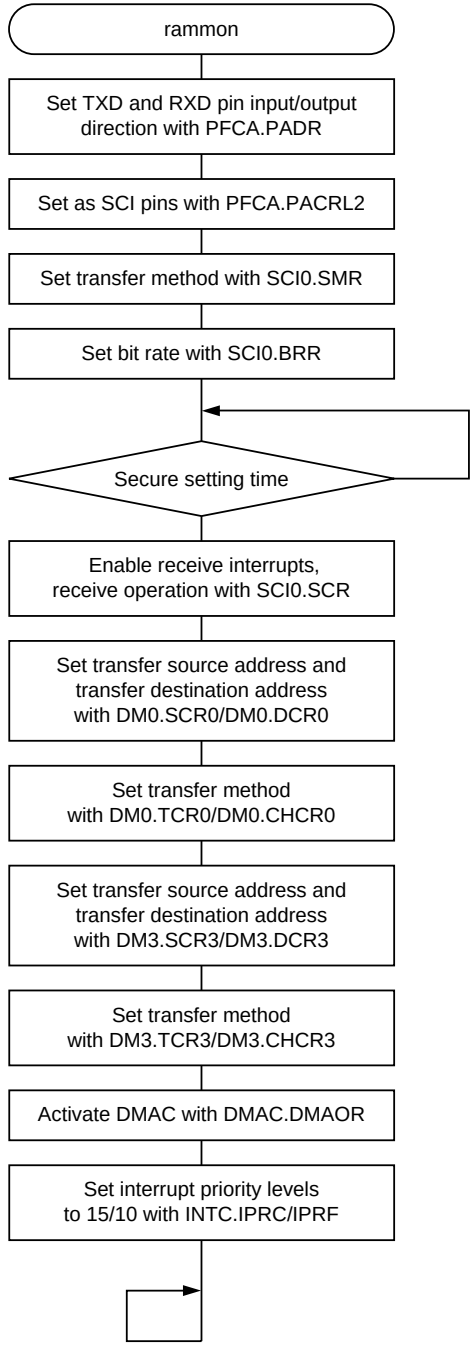
4. RAM Used

Label	Function	Data Length	Module
lk_addr	Stores RAM reference address	Unsigned long	Main routine

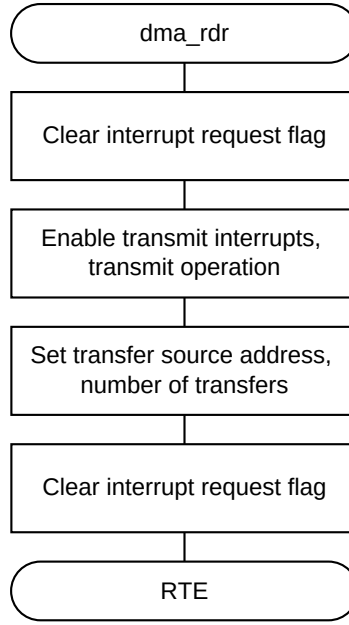
Note: The SH7040 header file name is used as the register label.

Flowchart

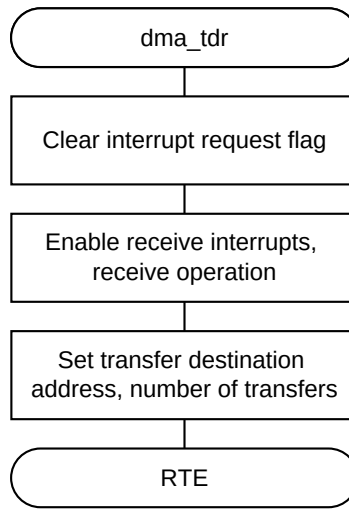
1. Main routine



2. Receive data transfer



3. Transmit data transfer



Program List

```
/*---------------------------------------------------------------------*/
/*                          INCLUDE FILE                               */
/*---------------------------------------------------------------------*/
#include<machine.h>
#include"sh7040.h"
/*---------------------------------------------------------------------*/
/*                           PROTOTYPE                                 */
/*---------------------------------------------------------------------*/
void rammon(void);
#pragma interrupt(dma0)
#pragma interrupt(dma_sci)
#pragma interrupt(sci_rxi)
/*---------------------------------------------------------------------*/
/*                         RAM ALLOCATION                              */
/*---------------------------------------------------------------------*/
#define data0(*(volatile unsigned char *)0xffffe800)
volatile struct addr
{
    long  addr0;                        /* transmit address */
};
#define dat(*(struct addr *)0xFFFFE810)

/*---------------------------------------------------------------------*/
/*                          MAIN PROGRAM                               */
/*---------------------------------------------------------------------*/
rammon()
{
    signed int lp;

    dat.addr0 = (long)&data0;
    data0 = 'H';
    PFCA.PADRL = 0x0002;                /* out put TXD0,input RXD0 */
    PFCA.PACRL2 = 0x0005;               /* TXD0,RXD0 sellect */
    SCI0.SCR0 = 0x00;                   /* stop transmit•receive */
    SCI0.SMR0 = 0x00;                   /* asynchronous mode,8bit data,not parity */
    SCI0.BRR0 = 0x40;                   /* bit rates 9600bps */
    SCI0.SCR0 = 0x00;                   /* clock seiiect */
    for(lp = 1; lp<1; lp++);            /* wait400ns */
    SCI0.SCR0 = 0x50;                   /* interrupt receive sci,receiving enabled */
    DM0.SAR0 = (long)(&SCI0.RDR);       /* source address:RAM */
    DM0.DAR0 = (long)(&dat.addr0);      /* distination address:SCI transmit register */
    DM0.TCR0 = 0x04;                    /* transmit count:4 */
    DM0.CHCR0 = 0x00004905;             /* indirect,source increment,byte-size transfer */
    DM3.SAR3 = (long)(&dat.addr0);      /* source address:RAM */
    DM3.DAR3 = (long)(&SCI0.TDR);       /* distination address:SCI transmit register */
    DM3.CHCR3 = 0x00181804;             /* indirect,source increment,byte-size transfer */
    DMAC.DMAOR = 0x0001;                /* data transfer enabled */
    INTC.IPRC = 0xe00f;                 /* set interrupt level=15 */
    INTC.IPRF = 0x00a0;                 /* set interrupt level=10 */
    set_imask(0x0);                     /* interrupt requested enabled */
    while(1);                           /* loop */
}
```

```
void dma_rdr(void)
{
    DM0.CHCR0 &= 0xfffffffd;             /* clear flag */
    SCI0.SCR0 = 0xa0;                    /* interrupt transmit sci,transmit enabled */
    DM3.SAR3 = (long)(&dat.addr0);       /* source address:RAM */
    DM3.TCR3 = 0x01;                     /* transmit count:4 */
    DM3.CHCR3 I= 0x00000001;             /* clear flag */
}

void dma_tdr(void)
{
    DM3.CHCR3 &= 0xfffffffc;             /* clear flag */
    SCI0.SCR0 = 0x70;                    /* interrupt receive sci,receiving enabled */
    DM0.DAR0 = (long)(&dat.addr0);       /* distination address:SCI transmit register */
    DM0.TCR0 = 0x04;                     /* transmit count:4 */
}
```

Appendix A

A.1 Header File

Program List

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/* Internal I/O register address define                                */
/*                                                                     */
/*---------------------------------------------------------------------*/

/*---------------------------------------------------------------------*/
/* INTC                                                                */
/*---------------------------------------------------------------------*/
volatile struct intc
{
        short IPRA;                     /* interrupt priority register A */
        short IPRB;                     /* interrupt priority register B */
        short IPRC;                     /* interrupt priority register C */
        short IPRD;                     /* interrupt priority register D */
        short IPRE;                     /* interrupt priority register E */
        short IPRF;                     /* interrupt priority register F */
        short IPRG;                     /* interrupt priority register G */
        short IPRH;                     /* interrupt priority register H */
        short ICR;                      /* interrupt controll register */
        short ISR;                      /* IRQ status register */
};

#define INTC (*(volatile struct intc *)0xffff8348)

/*---------------------------------------------------------------------*/
/*UBC                                                                  */
/*---------------------------------------------------------------------*/
volatile struct ubc
{
        short UBARH;                    /* user break address register H */
        short UBARL;                    /* user break address register L */
        short UBAMRH;                   /* user break mask register H */
        short UBAMRL;                   /* user break mask register L */
        short UBBR;                     /* user break bus cycle register */
};

#define UBC (*(volatile struct ubc *)0xffff8600)
```

```
/*------------------------------------------------------------------------*/
/* DTC                                                                    */
/*------------------------------------------------------------------------*/
volatile struct dtcsh
{
        char  DTEA;                     /* enable register A */
        char  DTEB;                     /* enable register B */
        char  DTEC;                     /* enable register C */
        char  DTED;                     /* enable register D */
        char  DTEE;                     /* enable register E */
        char  res1;
        short DTCSR;                    /* control/status register */
        short DTBR;                     /* information base register */
};

#define DTC (*(volatile struct dtcsh *)0xffff8700)

/*------------------------------------------------------------------------*/
/* CAC                                                                    */
/*------------------------------------------------------------------------*/
volatile struct cac
{
        char  CCR;                      /* cash control register */
};

#define CAC (*(volatile struct cac *)0xffff8740)

/*------------------------------------------------------------------------*/
/* BSC                                                                    */
/*------------------------------------------------------------------------*/
volatile struct bsc
{
        short BCR1;                     /* bus control register 1 */
        short BCR2;                     /* bus control register 2 */
        short WCR1;                     /* wait control register 1 */
        short WCR2;                     /* wait control register 2 */
        short res2;
        short DCR;                      /* DRAM area control register */
        short RTCSR;                    /* refresh timer control register */
        short RTCNT;                    /* refresh timer counter */
        short RTCOR;                    /* refresh timer constant register */
};

#define BSC (*(volatile struct bsc *)0xffff8620)
```

```
/*----------------------------------------------------------------------*/
/* DMAC                                                                 */
/*----------------------------------------------------------------------*/
volatile struct dmacsh
{
        long  DMAOR;                /* operation register */
};

volatile struct dmac0
{
        long  SAR0;                 /* source address register 0 */
        long  DAR0;                 /* destination address register 0 */
        long  TCR0;                 /* transfer count register 0 */
        long  CHCR0;                /* channel control register 0 */
};

volatile struct dmac1
{
        long  SAR1;                 /* source address register 1 */
        long  DAR1;                 /* destination address register 1 */
        long  TCR1;                 /* transfer count register 1 */
        long  CHCR1;                /* channel control register 1 */
};

volatile struct dmac2
{
        long  SAR2;                 /* source address register 2 */
        long  DAR2;                 /* destination address register 2 */
        long  TCR2;                 /* transfer count register 2 */
        long  CHCR2;                /* channel control register 2 */
};

volatile struct dmac3
{
        long  SAR3;                 /* source address register 3 */
        long  DAR3;                 /* destination address register 3 */
        long  TCR3;                 /* transfer count register 3 */
        long  CHCR3;                /* channel control register 3 */
};

#define DMAC (*(volatile struct dmac *)0xffff86b0)
#define DM0 (*(volatile struct dmac0 *)0xffff86c0)
#define DM1 (*(volatile struct dmac1 *)0xffff86d0)
#define DM2 (*(volatile struct dmac2 *)0xffff86e0)
#define DM3 (*(volatile struct dmac3 *)0xffff86f0)

/*----------------------------------------------------------------------*/
/* MTU                                                                  */
/*----------------------------------------------------------------------*/
volatile struct tmrsh1
{
        char  TSTR;                 /* timer start register */
        char  TSYR;                 /* timer synchronous register */
};

#define TMRSH (*(volatile struct tmrsh1*)0xffff8240)
/*----------------------------------------------------------------------*/
```

```
/* MTU CHO                                                              */
/*----------------------------------------------------------------------*/
volatile struct timer0
{
        char  TCR0;                     /* timer controll register 0 */
        char  TMDR0;                    /* timer mode register 0 */
        char  TIOR0H;                   /* timer i/o controll register 0H */
        char  TIOR0L;                   /* timer i/o controll register 0L */
        char  TIER0;                    /* timer interrupt enable register 0 */
        char  TSR0;                     /* timer stasus register 0 */
        short TCNT0;                    /* timer counter 0 */
        short TGR0A;                    /* timer general register 0A */
        short TGR0B;                    /* timer general register 0B */
        short TGR0C;                    /* timer general register 0C */
        short TGR0D;                    /* timer general register 0D */
};

#define T0 (*(volatile struct timer0 *)0xffff8260)

/*----------------------------------------------------------------------*/
/* MTU CH1                                                              */
/*----------------------------------------------------------------------*/
volatile struct timer1
{
        char  TCR1;                     /* timer controll register 1 */
        char  TMDR1;                    /* timer mode register 1 */
        char  TIOR1;                    /* timer i/o controll register 1H */
        char  res3;
        char  TIER1;                    /* timer interrupt enable register 1 */
        char  TSR1;                     /* timer status register 1 */
        short TCNT1;                    /* timer counter 1 */
        short TGR1A;                    /* timer general register 1A */
        short TGR1B;                    /* timer general register 1B */
};

#define T1 (*(volatile struct timer1 *)0xffff8280)

/*----------------------------------------------------------------------*/
/* MTU CH2                                                              */
/*----------------------------------------------------------------------*/
volatile struct timer2
{
        char  TCR2;                     /* timer controll register 2 */
        char  TMDR2;                    /* timer mode register 2 */
        char  TIOR2;                    /* timer i/o controll register 2H */
        char  res4;
        char  TIER2;                    /* timer interrupt enable register 2 */
        char  TSR2;                     /* timer status register 2 */
        short TCNT2;                    /* timer counter 2 */
        short TGR2A;                    /* timer general register 2A */
        short TGR2B;                    /* timer general register 2B */
};

#define T2 (*(volatile struct timer2 *)0xffff82a0)

/*----------------------------------------------------------------------*/
/* MTU CH3,4                                                            */
```

```
/*---------------------------------------------------------------------*/
volatile struct timer3
{
        char  TCR3;                     /* timer contorol register 3 */
        char  TCR4;                     /* timer contorol register 4 */
        char  TMDR3;                    /* timer mode register 3 */
        char  TMDR4;                    /* timer mode register 4 */
        char  TIOR3H;                   /* timer i/o contorol register 3H */
        char  TIOR3L;                   /* timer i/o contorol register 3L */
        char  TIOR4H;                   /* timer i/o contorol register 4H */
        char  TIOR4L;                   /* timer i/o contorol register 4L */
        char  TIER3;                    /* timer interrupt enable register 3 */
        char  TIER4;                    /* timer interrupt enable register 4 */
        char  TOER;                     /* timer output master enable register */
        char  TOCR;                     /* timer output contorol register */
        char  res5;                     /*
        char  TGCR;                     /* timer gate contorol register */
};
volatile struct timer31
{
        short TCNT3;                    /* timer counter 3 */
        short TCNT4;                    /* timer counter 4 */
        short TCDR;                     /* timer cycle-data register */
        short TDDR;                     /* timer deadtime-data register */
        short TGR3A;                    /* timer general register 3A */
        short TGR3B;                    /* timer general register 3B */
        short TGR4A;                    /* timer general register 4A */
        short TGR4B;                    /* timer general register 4B */
        short TCNTS;                    /* timer sub-counter */
        short TCBR;                     /* timer cycle buffer register */
        short TGR3C;                    /* timer general register 3C */
        short TGR3D;                    /* timer general register 3D */
        short TGR4C;                    /* timer general register 4C */
        short TGR4D;                    /* timer general register 4D */
        char  TSR3;                     /* timer status register 3 */
        char  TSR4;                     /* timer status register 4 */
};

#define T3(*(volatile struct timer3 *)0xffff8200)
#define T31(*(volatile struct timer31 *)0xffff8210)

/*---------------------------------------------------------------------*/
/* WDT                                                                 */
/*---------------------------------------------------------------------*/
volatile struct wdt
{
        char  TCSR;                     /* timer contorol status register */
        char  TCNT;                     /* timer counter */
        char  RSTCSR;                   /* riset contorol status register */
};

#define WDT(*(volatile struct wdt *)0xffff8610)
```

```
/*------------------------------------------------------------------------*/
/* SCI                                                                    */
/*------------------------------------------------------------------------*/
volatile struct sci0
{
        char  SMR0;                     /* serial mode register 0 */
        char  BRR0;                     /* bit-rate register 0 */
        char  SCR0;                     /* sirial contorol register 0 */
        char  TDR0;                     /* transmit data register 0 */
        char  SSR0;                     /* sirial status register 0 */
        char  RDR0;                     /* receive data register 0 */
};

volatile struct sci1
{
        char  SMR1;                     /* serial mode register 1 */
        char  BRR1;                     /* bit-rate register 1 */
        char  SCR1;                     /* sirial contorol register 1 */
        char  TDR1;                     /* transmit data register 1 */
        char  SSR1;                     /* sirial status register 1 */
        char  RDR1;                     /* receive data register 1 */
};

#define SCI0 (*(volatile struct sci0 *)0xffff81a0)
#define SCI1 (*(volatile struct sci1 *)0xffff81b0)

/*------------------------------------------------------------------------*/
/* A/D                                                                    */
/*------------------------------------------------------------------------*/
volatile struct a_d
{
        char  ADCSR;                    /* A/D contorol status register */
        char  ADCR;                     /* A/D contorol register */
};

volatile struct a_d0
{
        short ADDRA;                    /* A/D data register A */
        short ADDRB;                    /* A/D data register B */
        short ADDRC;                    /* A/D data register C */
        short ADDRD;                    /* A/D data register D */
        short ADDRE;                    /* A/D data register E */
        short ADDRF;                    /* A/D data register F */
        short ADDRG;                    /* A/D data register G */
        short ADDRH;                    /* A/D data register H */
};

#define A_D (*(volatile struct a_d *)0xffff83e0)
#define A_D0 (*(volatile struct a_d0 *)0xffff83f0)
```

```
/*-----------------------------------------------------------------------*/
/* CMT                                                                   */
/*-----------------------------------------------------------------------*/
volatile struct cmt
{
        short CMSTR;              /* compare/match timer start register */
        short CMCSR0;             /* compare/match timer contorol register 0 */
        short CMCNT0;             /* compare/match counter 0 */
        short CMCOR0;             /* compare/match constant register 0 */
        short CMCSR1;             /* compare/match timer contotol register */
        short CMCNT1;             /* compare/match counter 1 */
        short CMCOR1;             /* compare/match timer contorol register 1 */
};

#define CMT (*(volatile struct cmt *)0xffff83d0)

/*-----------------------------------------------------------------------*/
/* PFC                                                                   */
/*-----------------------------------------------------------------------*/
volatile struct pfca
{
        short PADRH;              /* portA data register H */
        short PADRL;              /* portA data register L */
        short PAIORH;             /* portA I/O register H */
        short PAIORL;             /* portA I/O register L */
        short PACRH;              /* portA contorol register H */
        short res6;
        short PACRL1;             /* portA contorol register L1 */
        short PACRL2;             /* portA contorol register L2 */
};
volatile struct pfcb
{
        short PBDR;               /* portB data register */
        short PCDR;               /* portC data register */
        short PBIOR;              /* portB I/O register */
        short PCIOR;              /* portC I/O register */
        short PBCR1;              /* portB contorol register 1 */
        short PBCR2;              /* portB contorol register 2 */
        short PCCR;               /* portC contorol register */
};
volatile struct pfcd
{
        short PDDRH;              /* portD data register H */
        short PDDRL;              /* portD data register L */
        short PDIORH;             /* portD I/O register H */
        short PDIORL;             /* portD I/O register L */
        short PDCRH1;             /* portD contorol register H1 */
        short PDCRH2;             /* portD contorol register H2 */
        short PDCRL;              /* portD contorol register L */
};
volatile struct pfce
{
        short PEDR;               /* portE data register H */
        char  PFDR;               /* portF data register */
        char  res7;
        short PEIOR;              /* portE I/O register */
        short res8;
```

```
        short PECR1;              /* portE contorol register 1 */
        short PECR2;              /* portE contorol register 2 */
};
volatile struct pfc0
{
        short IFCR;               /* IRQOUT contorol register */
};

#define PFCA (*(volatile struct pfca *)0xffff8380)
#define PFCB (*(volatile struct pfcb *)0xffff8390)
#define PFCD (*(volatile struct pfcd *)0xffff83A0)
#define PFCE (*(volatile struct pfce *)0xffff83B0)
#define PFC0 (*(volatile struct pfc0 *)0xffff83C8)

/*------------------------------------------------------------------------*/
/* POE                                                                    */
/*------------------------------------------------------------------------*/
volatile struct poe
{
        short ICSR;               /* input-level contorol status register */
        short OCSR;               /* output-level contorol status register */
};

#define POE (*(volatile struct poe *)0xffff83c0)
```

SH7040 Series On-Chip Supporting Modules Application Note

Publication Date: 1st Edition, September 1996
Published by: Semiconductor and IC Div.
Hitachi, Ltd.
Edited by: Technical Documentation Center
Hitachi Microcomputer System Ltd.